National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

NL-339 (r. 88/04)

Canada

THE UNIVERSITY OF ALBERTA

Classification of Data Structures for Thematic Data

by

© Luca Vanzella

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

Department of Computing Science

Edmonton, Alberta

Fall, 1988

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-45707-4

THE UNIVERSITY OF ALBERTA

Release Form

Name of Author: Luca Vanzella

Title of Thesis: *Classification of Data Structures for Thematic Data*

Degree for Which Thesis was Presented: Master of Science

Year This Degree Granted: 1988

Permission is hereby granted to **The University of Alberta Library** to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) Luca Vanzella

Permanent Address:

#1701, 10135 - Saskatchewan Drive
Edmonton, Alberta, Canada T6E 4Y9

Date: July 21, 1988

THE UNIVERSITY OF ALBERTA

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled *Classification of Data Structures for Thematic Data* submitted by **Luca Vanzella** in partial fulfillment of the requirements for the degree of **Master of Science.**

Supervisor

Wayne A. Davis

Date 14 June 1988

Abstract

Two fundamental data models are normally identified when classifying structures for the representation of spatial data. One model is based on the specification of locations and then recording attribute information for each location (i.e., tessellation or raster data structures). The second model is based on the specification of attributes and then recording locations for each attribute (e.g., vector data structures). It is generally claimed that, while tessellation structures are superior to vector structures with respect to performance of spatial operations, they are more storage demanding. This is known classically as the *storage versus performance* tradeoff between tessellation and vector structures. While this tradeoff is generally true for location-based operations, it is observed in this thesis that the second model is superior for attribute-based operations.

The tradeoffs between tessellation and vector models have long been the catalyst for the development of data structures that attempt to capitalize on the advantages of each model while minimizing the disadvantages. Consequently, a number of structures have appeared that, having properties of both fundamental models, cannot be accordingly classified.

Primarily for the purposes of classification of spatial data structures, we introduce a general *hybrid spatial data model*. The hybrid model consists of a tessellation structure which subdivides a vector encoded thematic data set into a set of adjacent cells. The tessellation component serves both as a partition of and as an index to the vector data within each cell.

The strength of the hybrid model is that it permits the classification of almost all existing spatial data structures. Limiting cases of the hybrid model correspond to tessellation and vector models. Intermediate formulations of the hybrid model yield most of the specific hybrid data structures developed in the literature. Realizations of certain

intermediate formulations can provide superior practical alternatives for representing large thematic data sets.

The classification of data structures is focused on the representation of thematic data in Geographic Information Systems. We are interested in very large thematic applications, i.e., where the data must reside in secondary storage. The cost of retrieving thematic data, by location or by attribute, is the underlying basis of evaluating data structures.

Acknowledgements

The author would like to thank the people who helped make this work possible. As with any research project, numerous people were consulted at all stages of its development. Their time and often insightful criticism of this work sparked new ideas and made old ones clear. In particular, I would like to thank the following people.

My first acknowledgements must go to my supervisor Stan Cabay. During the time I spent under his supervision I grew both academically and professionally. Stan's patience and understanding allowed my work to progress further than I could have expected. May everyone have such a helpful supervisor.

Second, I would like to thank the members of my examining committee, Dr. J. Culberson, Dr. W.A. Davis, and Dr. V. Noronha, for their time spent in reviewing this thesis.

Without the resources, technical support, and financial support provided by The Department of Computing Science at the University of Alberta, this thesis would not have been possible. The financial support of the Land Information Services Division (formerly the Alberta Bureau of Surveying and Mapping) of the Province of Alberta was also instrumental for the realization of this thesis.

I would like to thank James W.F. Smith for helping me initially to wade through the enormous sea of literature in this field and for sharing his technical knowledge of ARC/INFO. Finally, none of this would have been possible without the unlimited patience and kindness of Kathie Montegary and the ever present support of my parents.

Table of Contents

Chapter Page

Chapter 1: Introduction 1

1.1. Spatial Data 1

1.2. Spatial Operations 4

1.2.1. Location-Based Operations 4

1.2.2. Attribute-Based Operations 5

1.2.3. Thematic Overlay 7

1.3. Spatial Data Structures 8

1.4. Other Surveys 11

Chapter 2: Location-Based Data Structures 14

2.1. Introduction 14

2.2. Pointer-Based Quadtrees 18

2.3. Pointerless Quadtrees 25

2.4. Cell Methods 29

2.5. Location-Based Operations 42

2.6. Attribute-Based Operations 45

2.7. Thematic Overlay 49

Chapter 3: Attribute-Based Data Structures 51

3.1. Introduction 51

3.2. Polygon-Driven Structures 53

3.3. Curve-Driven Structures 57

3.4. Attribute-Based Operations 62

3.5. Location-Based Operations 67

3.6. Thematic Overlay 73

Chapter 4: Hybrid Data Structures 74

4.1. Introduction 74

4.2. A Hybrid Spatial Data Model 75

4.2.1. Tessellation Component 77

4.2.2. Vector Component 80

4.2.3. Resolution Thresholds 82

4.2.4. Updating The Hybrid Data Structure 98

4.2.5. Constructing The Hybrid Data Structure 100

4.3. Spatial Operations 101

4.3.1. Location-Based Operations 105

4.3.2. Attribute-Based Operations 107

4.3.3. Thematic Overlay 109

4.3.4. Summary 113

4.4. Tile Methods 117

4.5. Other Hybrid Structures 125

Chapter 5: Concluding Remarks 129

5.1. Summary 129

5.2. Future Perspectives 131

References 135

A1: Storage Requirement for the Hybrid Data Structure 143

List of Tables

Table Page
4.1 Summary of Retrieval and Processing Costs. 116

A1.1 Characteristics of the IBM 3380K Device. 144

A1.2 Storage Requirement for a Tessellation Component Node. 144

List of Figures

Figure Page
1.1 A Polygon Network. 3
1.2 Windowing Operation. 5
1.3 Polygon Set Operations. 6
1.4 Thematic Overlay Operation. 7
2.1 Array Representation of a Polygon Network. 16
2.2 Run Length Encoding. 17
2.3 Region Quadtree Representation. 19
2.4 Field Tree Decomposition Principle. 22
2.5 Field Tree Representation. 23
2.6 Linear Quadtree Representation. 27
2.7 DF-expression. 28
2.8 Extendible Hashing Representing a Linear Quadtree. 33
2.9 Linear Hashing Representing a Linear Quadtree. 36
2.10 Linear Quadtree of a Polygon Window. 44
2.11 Forest of Quadtrees. 48
3.1 Vector Representation of a Polygon Network. 52
3.2 Eight Direction Chaincodes. 52
3.3 Curve-Driven Data Structure. 58
3.4 POLYVRT Data Structure. 59
3.5 Weiler's Data Structure.[2] 64

3.6 R-Tree Data Structure. 69

4.1 MX Quadtree Representation of a Polygon.[4] 86

4.2 Line Quadtree for the Polygon Network of Figure 1.1. 89

4.3 Example of Vertex Merging. 91

4.4 A Polygon Network and its PM Quadtree.[5] 92

4.5 Example of Edge Merging. 94

4.6 Edge Quadtree for the Polygon of Figure 4.1.[6] 96

Chapter 1

Introduction

This thesis is concerned with the representation of thematic data. By representation, we mean data structures used in computer systems. In particular, the computer systems we are interested in are *Geographic Information Systems* (GIS) that are used in thematic applications. The purpose of this research is to investigate, classify, and evaluate data structures for the effective representation and manipulation of thematic data.

Throughout the thesis, the evaluation of data structures is based on the cost of performing a set of operations applied to thematic data. We call these operations *spatial operations*. An underlying assumption of the thesis is that we are dealing with very large thematic data sets that consequently must reside on secondary (external) storage. Thus, our analysis and evaluation is primarily based on the cost of accessing thematic data needed for spatial operations. The cost of processing thematic data in primary storage (memory) is given only secondary consideration.

1.1. Spatial Data

We begin by defining what we mean by spatial data in general and then we describe the type of data that this thesis focuses on: thematic data. Spatial data can be defined as any data concerning phenomena spatially distributed in one or more dimensions [68]. This is a large class of data and is often referred to as pictorial data (e.g., [14]). This thesis is concerned with spatial data pertaining to the surface of the Earth which is commonly represented by two-dimensional models known as thematic maps. Our interest is focused on thematic data, that is, data found on thematic maps that describes some qualitative or quantitative phenomenon (theme) across an area of interest. Examples of what we mean by the terms *thematic data* and *theme* include forest covers, soil types, and land use classifications. Generally, a theme partitions an area of interest into a collection of regions commonly termed *polygons*. We use the term *polygon network* to mean the collection of polygons comprising a theme. In particular, a polygon network is a set of *edges* connected at their vertices to form a collection of non-overlapping regions that totally partitions an area of interest. We assume that edges are straight line segments and that vertices are represented by 2-dimensional coordinates. The regions bounded by the set of edges are commonly termed *polygons*, of which there are three types. *Simple* polygons are regions without holes (i.e., internal polygons) while *complex* polygons are regions containing one or more holes. *Compound* polygons refer to the union of two or more simple or complex polygons.

For the purposes of this thesis, a polygon network is equivalent to a layer of thematic information covering an area of interest. Generally, themes are defined in terms of a set of attributes that serves to both classify the data and define the polygons of the thematic layer. Each polygon can be identified or labelled by the attribute value that defines it. An attribute value such as *green* may actually refer to several disjoint regions within a polygon network (i.e., a compound polygon).

Figure 1.1 shows a polygon network describing a fictitious theme. All three types of polygons are present within the network. The colors of the theme are labelled A, B, C, and D. A and D refer to simple polygons, C refers to a complex polygon, and B refers to a compound polygon.

Three basic data types - points, lines, and areas - can be used to describe any polygon. By dealing with thematic data, this thesis considers all three data types collectively. Individual consideration of point and line data is excluded from this thesis. Other, more appropriate data structures, have been developed in the literature that deal expressly with point and line data. We note that the representation of thematic data is a

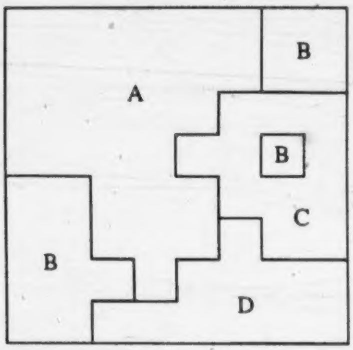


Figure 1.1 A Polygon Network.

more complex issue than the representation of either point or line data exclusively.

For the purposes of this thesis, we divide thematic data into two broad categories: *location data* and *attribute data* . Location data refers to a coordinate system that serves as a reference for the positioning of attribute data. For positioning on the surface of the Earth, spherical coordinates (latitudes and longitudes) are normally used. Two-dimensional thematic maps require projecting a sphere onto a plane using some coordinate transformation. Many types of plane coordinate systems are used for positioning on a map (e.g., conformal, transverse mercator, conic, or any of the widely used rectilinear grid systems). The location and shape of a polygon is the information that is encoded in location data. It consists of sets of coordinates that explicitly define points which, in turn, implicitly define lines and areas.

Attribute data is the non-positional, descriptive information associated with the polygons displayed on thematic maps. Attribute data provides either nominal (textual) or scalar (numeric) information about polygons. A polygon can have several attributes associated with it but there is always one attribute that is responsible for its geometric description. We refer to this primary attribute as *color*.

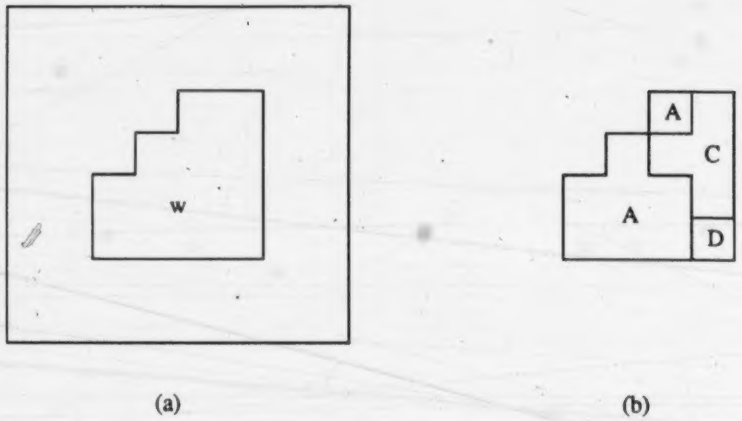
1.2. Spatial Operations

GIS are systems that are used to store, manipulate, and display spatial data. Although our emphasis of GIS is on thematic data, any type of data may be associated with positions and processed in a GIS. Manipulation of thematic data includes the application of *spatial operations*. There are two classes of spatial operations which are characterized by the type of access required to retrieve the relevant thematic data from one or more polygon networks. These classes are *attribute-based* operations and *location-based* operations. Under each class in this thesis, we restrict ourselves to a set of operations that we consider most relevant to GIS's used in thematic applications. While this set of operations is not exhaustive, it is representative of the two fundamental ways of describing and searching thematic data.

1.2.1. Location-Based Operations

The class of location-based operations is characterized by the necessity of search for specific locations within a polygon network. That is, given a location, determine the attribute(s) corresponding to or intersecting the location. The windowing operation and the point inclusion operation are examples of location-based operations. Both require search for specific locations in space and both return a set of polygons, hence attributes, which intersect the locations.

The windowing operation consists of extracting a portion of a polygon network, n, that is contained within a window, w. In general, the window is defined as a polygon. Windowing is performed by intersecting the polygons in n with w. This operation is also known as *range search* or *clipping*. Figure 1.2 shows an example of windowing. Figure 1.2a shows a polygon representing a window and Figure 1.2b depicts the result of extracting the window from the polygon network of Figure 1.1.



(a) (b)

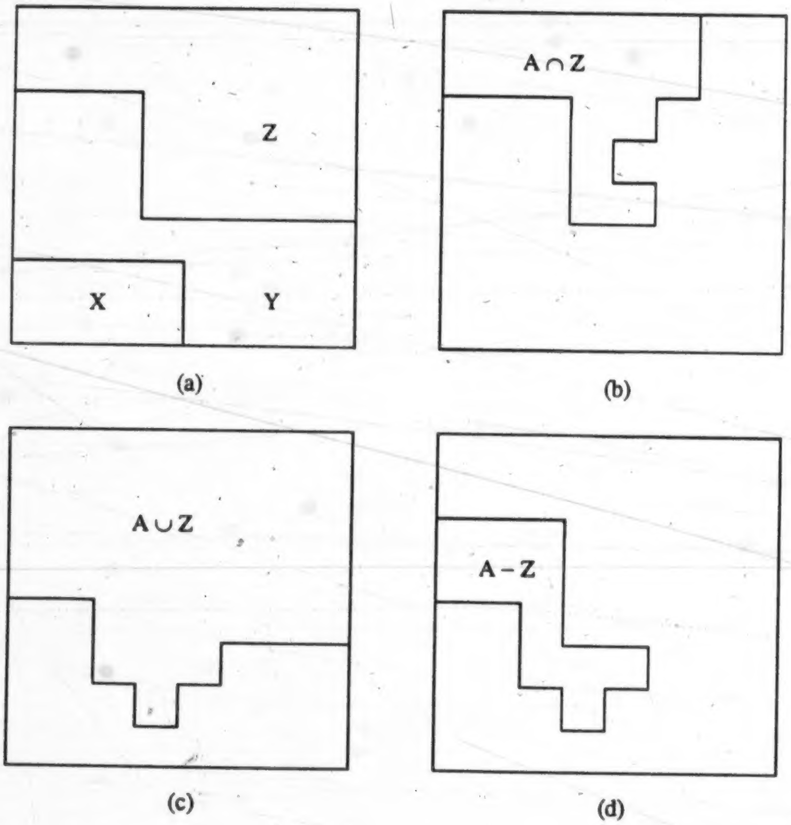
Figure 1.2 Windowing Operation.

The point inclusion operation is a special case of the windowing operation where the window is specified as a point on the plane. Given a polygon network n and a point p, the point inclusion problem is to determine which polygon of n contains the point. The result of point inclusion is a description of the enclosing polygon, its color, or other data associated with the polygon. This operation is related to the *point-in-polygon* operation that determines whether a given polygon contains a point.

1.2.2. Attribute-Based Operations

The class of attribute-based operations is characterized by search for specific attribute values (colors) within a polygon network. That is, given an attribute value, determine the locations possessing that value. In a polygon network, these locations consist of polygons that possess specified colors. The polygon set operations are examples of attribute-based operations. Each operation involves finding those polygons that possess the specified colors and then performing the set operation. The resulting polygons define the locations of the combined attributes.

In general, the polygon set operations involve two polygon networks n_1 and n_2 and a color from each network c_1 and c_2. They all begin by locating all polygons possessing color c_1 as well as all polygons possessing color c_2. Figure 1.3 illustrates the usual polygon set operations of intersection, union, and difference. In Figure 1.3, these operations are illustrated using the polygon networks n_1 (given in Figure 1.1) and n_2 (given in Figure 1.3a). Figures 1.3b,c,d give the results of the intersection, union, and difference operations, respectively, involving A from n_1 and Z from n_2.



(a) (b)

(c) (d)

Figure 1.3 Polygon Set Operations.

1.2.3. Thematic Overlay

Probably the most important operation in GIS applications is that of thematic overlay. The thematic overlay operation involves two polygon networks n_1 and n_2 and consists of applying the intersection operation to every possible pair of polygons, one from each network. This operation obtains a polygon network n_3 such that $n_3 = n_1 \times n_2$, where × is the cartesian product of the set of polygons in n_1 and the set of polygons in n_2. Figure 1.4 shows the polygon network resulting from overlaying the polygon networks of Figures 1.1 and 1.3a. Each polygon in Figure 1.4 inherits a color corresponding to the colors of the intersecting polygons.

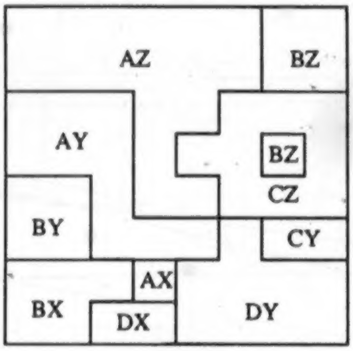


Figure 1.4 Thematic Overlay Operation.

In essence, thematic overlay is an intersection operation that is applied exhaustively at all locations and involves all attributes. It is not restricted by either specific locations or specific attributes. For this reason, thematic overlay is given a separate designation in this thesis, apart from location-based and attribute-based operations.

1.3. Spatial Data Structures

There are two major models for the representation of thematic data. In the first model, locations are specified and for each location relevant attributes are recorded. Conversely, in the second model, attributes are specified and for each attribute relevant locations are recorded.

The first model for thematic data representation is based on the idea of subdivision or *tessellation* of the plane into cells. While two classes of tessellations exist, regular and irregular, this thesis addresses only the first type. Each cell in a tessellation represents and defines a region that can be differentiated from an adjacent region. A common term for the smallest cell of a tessellation (i.e., a unit area) is *pixel*. Each cell identifies a location on the plane, typically by two-dimensional coordinates. For example, the center of the cell may be specified by the coordinates. For each cell, attributes are recorded. For thematic data, attributes correspond to colors. The data structures associated with the organization of thematic data in this manner are called *location-based data structures* in this thesis. Location-based data structures are the subject of Chapter 2. Other designations for these types of data structures are found in the literature: tessellation (e.g., [68]), cellular (e.g., [59]), and raster (e.g., [10]).

In the second model for thematic data representation, the locations of each attribute (i.e., color) are explicitly or implicitly recorded. The data structures associated with the organization of thematic data in this manner are called *attribute-based data structures* in this thesis. Attribute-based data structures can be realized in a number of different ways and are the subject of Chapter 3. The most primitive way is to list, for each color, all of the cells of that color. This method constitutes an inverted list of the first model for thematic data representation. The inverted list is as storage demanding as a tessellation approach since all locations are still explicitly recorded (i.e., inversion implies reorganization of the data, not reduction of the storage required). However, it is not necessary to list all locations explicitly for each attribute. For this reason, the inverted list method is not used in practice. A more common way of recording locations is to list the boundaries associated with each color. This can be done by listing the pixels along the boundary of each region. Typically, to save additional storage space, only the locations where the boundary changes direction are recorded. The pixels representing changes in direction are usually called *vertices*. Specifying a boundary by storing only vertices requires the imposition of an ordering on the vertices stored. Segments between successive vertices are called edges or *vectors*, hence methods of this type are frequently classified as vector data structures (e.g., [10, 68]). The usual class of vector data structures is a sub-class of our class of attribute-based data structures.

The foregoing discussion points to a duality between the two fundamental models for thematic data representation. Typically, this duality is recognized from the following observations [68]. Vector structures use a line segment as the basic unit of organization and line segments are the units for which location information is recorded. Tessellation structures use the unit area as the basic unit wherein attribute information is recorded for each unit area. In this thesis, the duality of the two models is more correctly shown to arise from the location-based versus attribute-based approaches to thematic data representation.

In Chapter 2, we show that location-based data structures are more cost effective for location-based operations. Conversely, in Chapter 3, we show that attribute-based data structures are more cost effective for attribute-based operations. Regarding thematic overlay, we claim that the location-based data structures are more cost effective. Mainly for the last reason above, the usual claim in the literature is that location-based (tessellation) data structures are superior to attribute-based (vector) data structures with respect to performance. It is also generally accepted, for reasons implied by the previous paragraph, that vector data structures are less storage demanding than tessellation data structures, especially for thematic data. These two considerations are the basis of the widely entrenched notion of the *storage versus performance tradeoff* between vector and tessellation data structures [10, 47, 51, 59, 68, 89, 90]. We wish to temper this notion with the realization that vector data structures are more effective for an important class of spatial operations, namely, the attribute-based operations.

Nevertheless, both classes of data structures have advantages. A number of data structures have been developed that attempt to capitalize on the strengths of both classes. These data structures possess aspects of both the location-based and the attribute-based approaches. We distinguish these structures by a third class, namely, *hybrid* data structures.

For the purpose of classifying hybrid data structures, in Chapter 4, we introduce a general hybrid model. The goal of the hybrid model is to capitalize on both the storage and retrieval advantages of vector and tessellation approaches while minimizing the tradeoffs. The hybrid model is based on the method of *divide-and-conquer* [2] applied to a vector encoded data structure. The hybrid model is described by introducing a tessellation to subdivide a polygon network into a set of adjacent cells. This tessellation component of the model serves both as a partition of and as an index to the polygon network. Within each cell, the thematic data is still represented by the vector component. The subdivision of the tessellation component is allowed to continue to any level of resolution. If the resolution is that of pixel size cells, then one limiting case of the model is achieved, namely, a raster structure. At the other extreme, if the resolution is limited to a single cell covering the entire polygon network, then the other limiting case is achieved, that is, a vector data structure.

The tessellation component of the hybrid model is the division aspect of the divide-and-conquer method. The conquer aspect is achieved by storing sufficient information in each cell to permit the independent retrieval and processing of thematic data for the spatial operations. The strengths of the hybrid model are two-fold. First, as noted above, limiting cases of the model correspond to the vector and tessellation approaches. More importantly, intermediate formulations of the model yield most specific hybrid data structures recently developed and described in the literature. Second, we claim that realizations of certain intermediate formulations of the hybrid model can provide superior, practical alternatives for representing thematic data in GIS's.

1.4. Other Surveys

Nagy and Wagle [59] discuss and compare spatial data structures in their survey on geographic data processing. Their classification is based on cellular (raster) and linked (vector) methods of spatial data organization. This survey does not include many of the structures and concepts covered in this thesis. Peucker and Chrisman [65] survey vector methods for 2-D and 3-D applications. Although this survey is restricted to vector structures, it does discuss a structure that has influenced the design of many subsequent vector methods.

Peuquet [68] proposes a framework for the discussion and comparison of many known spatial data structures. Her framework is divided into three classes of data models: tessellation, vector, and hybrid models. While the organization of her survey is similar to that of this thesis, there is no focus on performance issues and the class of hybrid data models is not well developed.

Burrough [10] reviews tessellation and vector data structures for thematic data in a chapter of his book dealing with the general principles of GIS. He deals with the usual raster methods including the region quadtree and a number of vector structures. His comparison of the raster and vector approaches includes a general discussion of the storage versus performance tradeoff, but does not focus on the location-based versus attribute-based approaches to searching thematic data.

Davis [21] surveys a number of different data formats for *image* (raster) and *graphics* (vector) data. He makes the important pragmatic point that spatial data should be stored in a format which closely resembles its source format. This is important for two reasons. Conversion between raster and vector can reduce the accuracy of the data. Furthermore, organizations maintaining spatial data may not want to convert their data. This second situation suggests that, for practical considerations, the question of raster versus vector is not always an issue; that is, there often is no choice about the representation type.

Samet's [77] tutorial survey on the quadtree and related hierarchical data structures is a comprehensive reference for this important class of representation techniques. Samet primarily focuses on the representation of region data which is closely related to thematic data. He deals extensively with tessellation structures and treats vector structures superficially. Samet deals also with the representation of point and line data, although not as extensively. Throughout his survey, Samet discusses the performance of a wide variety of spatial operations. Consequently, he does not focus on the spatial operations discussed in this thesis.

This thesis focuses on thematic data, a complex and representative subset of pictorial or spatial data. For surveys and discussions of data structures of two and higher dimensional pictorial data, the reader is referred to the following. Rosenfeld and Kak's [71] text on digital picture processing is a comprehensive reference in the area of pictorial representation. A survey of representation techniques for pictorial data is given by

Chang and Kunii [14]. Chang, et al [13] describe a system for the management of large amounts of pictorial information. The system is based on the relational approach to database design and uses a *logical picture* as the basic entity for storage and retrieval. A logical picture is defined as a hierarchically structured collection of picture objects that consist of *physical pictures* stored as either tessellation or vector data in an image store. Tanimoto [85] discusses several hierarchical indices that may be used for access to pictorial data. Examples of such indices are based on inherent picture structure or on picture contents. The performance of spatial operations on three-dimensional objects using methods related to the hybrid model of this thesis is described by Carlbom [12].

This thesis does not address levels of abstraction or organization higher than data structures in the area of spatial data representation. This includes consideration of issues in data modelling and database organization. While these considerations are important for the development of GIS's, they deal with issues of more abstract organization. Since these issues are important, however, we mention some representative literature. Lorie and Meier [50] discuss the use of a relational database as a geographical database. Frank [32] discusses the issues involved in the design of large GIS's. Rugg [74] describes the use of a Hypergraph-Based data structure for the hierarchical representation of diverse spatial data types in an integrated manner. The GIS conference literature annually contains many papers discussing the organization of data in GIS's at many levels of abstraction. A representative example from 1987 is the session on database architecture found in the proceedings of Auto Carto 8 [88].

Chapter 2

Location-Based Data Structures

2.1. Introduction

Location-based data structures are developed on the idea of subdivision (tessellation) of the plane by cells. The primary differences among structures in this class are the shape of the cell used, whether the tessellation is regular or irregular, and whether the tessellation is flat or nested. Fundamentally, tessellation structures use locations in space (corresponding to unit areas) as the basic logical units of information. The unit area is commonly known as a *pixel* and attribute information is recorded for each pixel.

A tessellation is a partition of the plane into a set of non-overlapping and collectively exhaustive cells. If k represents the number of sides of a cell and v represents the number of cells meeting at a vertex, then a (k,v)*–regular tessellation* is a partition in which the value of k is the same for all cells and v is the same for all vertices. This definition restricts the class of (k,v)-regular tessellations to three types: triangular, rectangular, and hexagonal [3]. By far the most useful tessellation for thematic applications is the rectangular tessellation.

Irregular tessellations include those in which k is fixed but v can vary from cell to cell (i.e., the cells can vary in size and density). The most common of these is probably the Triangulated Irregular Network (TIN), which has found a successful use in approximating 3-dimensional data such as terrain data. For 2-dimensional thematic data, however, the generation and maintenance of structures based on irregular tessellations is computationally expensive and operations such as overlay are difficult to perform, at best [68].

Kirkpatrick [44] develops a hierarchical decomposition method based on a triangular subdivision of a planar graph (a polygon, for our purposes). The triangulation process begins by enclosing an arbitrary polygon within a triangle. The regions interior to the polygon and exterior to the polygon (but interior to the triangle) are then triangulated. A triangulation hierarchy is constructed by successively replacing a group of triangles with a common vertex by a smaller group of triangles. The common vertex is eliminated and the resulting subdivision is retriangulated. This process continues until only the enclosing triangle remains. The hierarchical set of triangular subdivisions can be represented with a tree structure and is termed a *k-structure*.

Although the k-structure can be used for the representation of vector encoded polygon networks, it is a location-based method since it is based on the subdivision of regions. Since the boundary of a polygon governs the decomposition, the k-structure induces an irregular tessellation. The k-structure enables a subdivision search algorithm, which can be used for point-in-polygon operations, that is theoretically optimal for both search time and space requirement. Although theoretically elegant, the k-structure is definitely not a practical method for representing polygon networks since a change to one polygon requires retriangulation of the entire network. Samet [77] has also noted the unsuitability of the k-structure by commenting that a general updating procedure for this structure has not been reported.

Historically, the most widely used tessellation structure has been the flat, square grid commonly referred to as a *raster*. The main reasons for this are its compatibility with the array structure of the FORTRAN programming language and with devices used for spatial data capture (e.g., scanners) and raster-based output devices [68]. The simplest format for representing a polygon network in raster is that of a 2-d array with each array element storing the color of one pixel. Figure 2.1 shows the raster array representation for the polygon network of Figure 1.1.

A	A	A	A	A	A	B	B
A	A	A	A	A	A	B	B
A	A	A	A	A	C	C	C
A	A	A	A	C	C	B	C
B	B	A	A	A	C	C	C
B	B	A	A	A	D	C	C
B	B	B	A	D	D	D	D
B	B	D	D	D	D	D	D

Figure 2.1 Array Representation of a Polygon Network.

The primary drawback of a raster data format is the excessive storage required because all pixels are represented, regardless of color. Another serious limitation is that boundary information is difficult to extract.

For large thematic data sets represented by raster structures, it must be assumed that the array representation is located in contiguous external storage. This permits the use of direct access storage techniques and allows for the the efficient performance of the location-based operations. The point inclusion operation is simply an array lookup operation which can be performed with $O(1)$ accesses. Rectangular windowing involves computing the array elements corresponding to diagonally opposite corners and selecting only those pixels in between those locations [21]. On the other hand, the attribute-based operations require exhaustive search since there is no provision for referencing the locations of colors. This takes $O(p)$ retrievals, where p is the number of pixels in the array. The thematic overlay operation consists of traversing two registered array representations and comparing pairs of pixels. This operation also requires $O(p)$ retrievals.

The storage problem of the raster can be reduced by using a compression technique known as *run length encoding* [64]. Each row in the array representation may have consecutive pixels with the same color. Run length encoding compresses each row of pixels into a sequence of values denoting pixels of the same color. For example, a set of n consecutive pixels with color c may be denoted as the two values n, c. Figure 2.2 shows a run length encoding for the array representation of Figure 2.1. Run length encoding can result in significant space saving [64] and it does not adversely affect the performance of the location-based operations. It does, in fact, speed up the windowing, attribute-based, and overlay operations since fewer records are stored. However, the retrieval cost complexity for these operations remains $O(p)$, where p is the number of pixels. An example of a run length encoding method and its application to representing terrain data is described in [57].

Row	Run Length Encoding
1	6A, 2B
2	6A, 2B
3	5A, 3C
4	4A, 2C, 1B, 1C
5	2B, 3A, 3C
6	2B, 3A, 1D, 2C
7	3B, 1A, 4D
8	2B, 6D

Figure 2.2 Run Length Encoding.

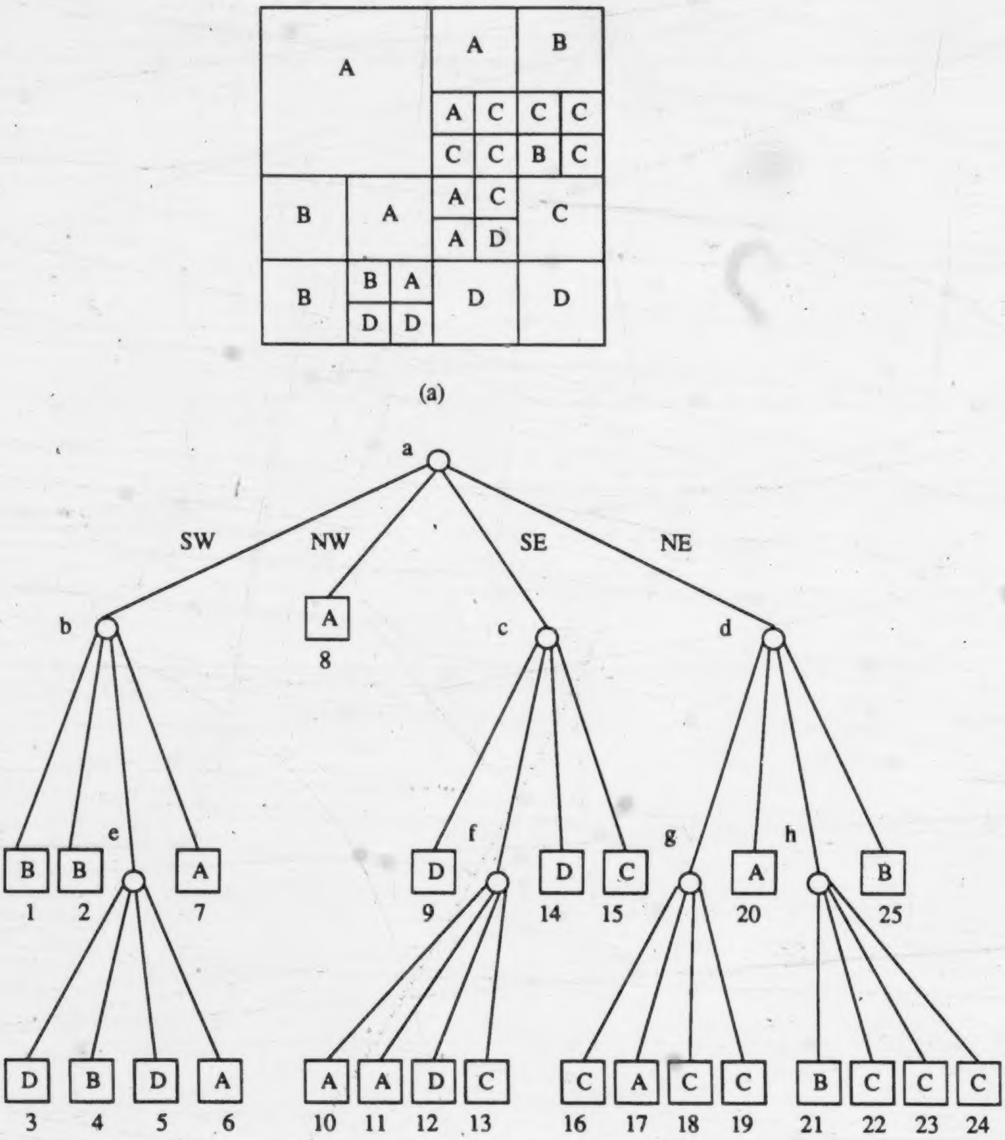
Another technique for reducing the storage requirement is the class of *nested tessellation models* or *hierarchical data structures* collectively known as *quadtrees*. Hierarchical data structures have become an increasingly important representation technique in the fields of GIS, image processing, and computational geometry [77]. As such, we review this class in the next section.

2.2. Pointer-Based Quadtrees

The term quadtree has come to mean a class of nested tessellation models whose common property is that they are based on the principle of recursive decomposition of space [77]. Quadtree structures are differentiated on the basis of the type of data represented and on the principle guiding the decomposition process. Quadtrees have been proposed for the representation of point, line, and region data. The *point quadtree* [77] is used to represent point data and is an example of a decomposition process governed by the input data. This results in an irregular decomposition of space.

Quadtrees used to represent region data are based on the regular decomposition of space, that is, they subdivide the embedding space in a regular manner, independently of the location of the data. The most studied quadtree approach to region representation is the *region quadtree* , which is based on a recursive, square tessellation. This method represents an image based on the successive decomposition of the image space into four quadrants of equal size. Quadrant subdivision continues until homogenous subquadrants (or blocks), possibly pixels, are obtained. The decomposition is usually represented by a tree structure of degree four. The nodes of the tree correspond to subquadrants of the image. In the case of a binary image, all the leaf nodes are either BLACK or WHITE and all interior nodes are said to be GRAY.

Polygon networks may be represented by region quadtrees as well. In this case quadrants are subdivided if they do not lie wholly within one polygon. Each of the leaf nodes for such a *multi-color* quadtree is assigned a color designation. This type of quadtree encoding is the basis for a GIS used to process thematic data such as land-use classifications [70, 72]. An example of a region quadtree is found in Figure 2.3. Figure 2.3a shows the maximal block decomposition of the polygon network of Figure 1.1. The tree representation is shown in Figure 2.3b.


A
A
B
A
C
C
C
C
C
B
C
B
A
A
C
C
A
D
B
B
A
D
D
D
D
a
SW
NW
SE
NE
b
A
8
c
d
e
f
g
h
B
1
B
2
A
7
D
9
D
14
C
15
A
20
B
25
D
3
B
4
D
5
A
6
A
10
A
11
D
12
C
13
C
16
A
17
C
18
C
19
B
21
C
22
C
23
C
24

(a)

(b)

Figure 2.3 Region Quadtree Representation.

The most natural way to represent quadtrees is to use a quaternary tree structure. Each record describes a quadtree node and has space for at least four pointers; one each for the four subquadrants. Spatial operations can then be implemented as tree traversal algorithms. In particular, the set operations, windowing and the overlay operation require exhaustive traversals of quadtree pairs in parallel. The point inclusion operation consists of following the path from the root of the tree to the leaf node containing the point, and requires $O(\log n)$ retrievals, where n is the number of nodes in the tree. With the quaternary formulation, a complete quadtree of height h has $(4^{h+1}-1)/3$ nodes, where the root node has height zero.

The region quadtree represents an improvement on the storage required by the raster due to the 2-dimensional aggregation of homogenous blocks of pixels (as opposed to the 1-dimensional compression of run length encoding). The improvement varies according to the level of aggregation in the data. In fact, for color maps of considerable detail or for unclassified satellite data, quadtrees may not be appropriate [21, 89]. This is because of the considerable storage overhead associated with tree representations. A quadtree with λ leaf nodes has $(\lambda-1)/3$ internal nodes. The storage of the pointers further increases the space requirement. The use of pointers leads to the additional problem of pointer following operations that are required when performing tree traversals. The latter problem is especially acute when quadtrees are kept in external storage, as pointer following can lead to an unacceptable number of disk accesses [24, 77].

One way to reduce the pointer storage overhead is to subdivide the space into two equal sized parts always switching between the x and y axes. This method, termed a *bintree* [46] is analogous to the multidimensional binary search tree (or *k-d tree*) proposed by Bentley [6]. Using a bintree reduces the number of pointers needed for each internal node to two and generally leads to fewer leaf nodes [77]. However, the height of a bintree for any study area is twice the height of the corresponding quadtree, which typically doubles the cost of location-based operations. The cost of the point inclusion operation, for example, requires $O(\log n)$ retrievals, where n is the number of nodes.

Frank [30] introduces a variation on the decomposition principle of the quadtree that is the basis of a structure called a *Field Tree* (see also [31]). The decomposition principle is altered so that each level of subquadrants is systematically shifted on the plane. In both the quadtree and the Field Tree, each cell is subdivided into a number of smaller, congruent cells that totally overlap the original cell. For the quadtree, the number is always four but for the Field Tree, since the smaller cells are shifted, usually nine cells are required. Figure 2.4 illustrates the decomposition principle for the first three levels of a Field Tree. Level zero consists of a single cell covering a study area. This cell, bounded by solid lines in Figure 2.4, corresponds to the root cell of a quadtree decomposition for the same area. The root cell is subdivided into cells 1/4 its size (like the quadtree) but these level one cells are translated in both x and y by 1/4 the side length of the root cell. To fully cover the root cell, nine level one cells (bounded by dashed lines in the Figure) are required. Each level one cell is again subdivided into nine cells, at level two, but some of these are shared by adjacent level one cells. Four level two cells (bounded by dotted lines in the Figure) are associated with a level one cell. The other five cells are associated with adjacent level one cells. Thus, level one of the Field Tree is subdivided into forty nine level two cells.

Figure 2.5 shows a partial tree realization of the Field Tree in Figure 2.4. Since a quaternary tree structure is used, only four descendents of a node are immediately below the node. The other five descendents are located in standard locations in the subtrees of the north, northeast, and east neighbors of the node. For example, the children of node 12 in Figure 2.5 are located in the subtrees of nodes 12, 28, 44, and 60. The five non-

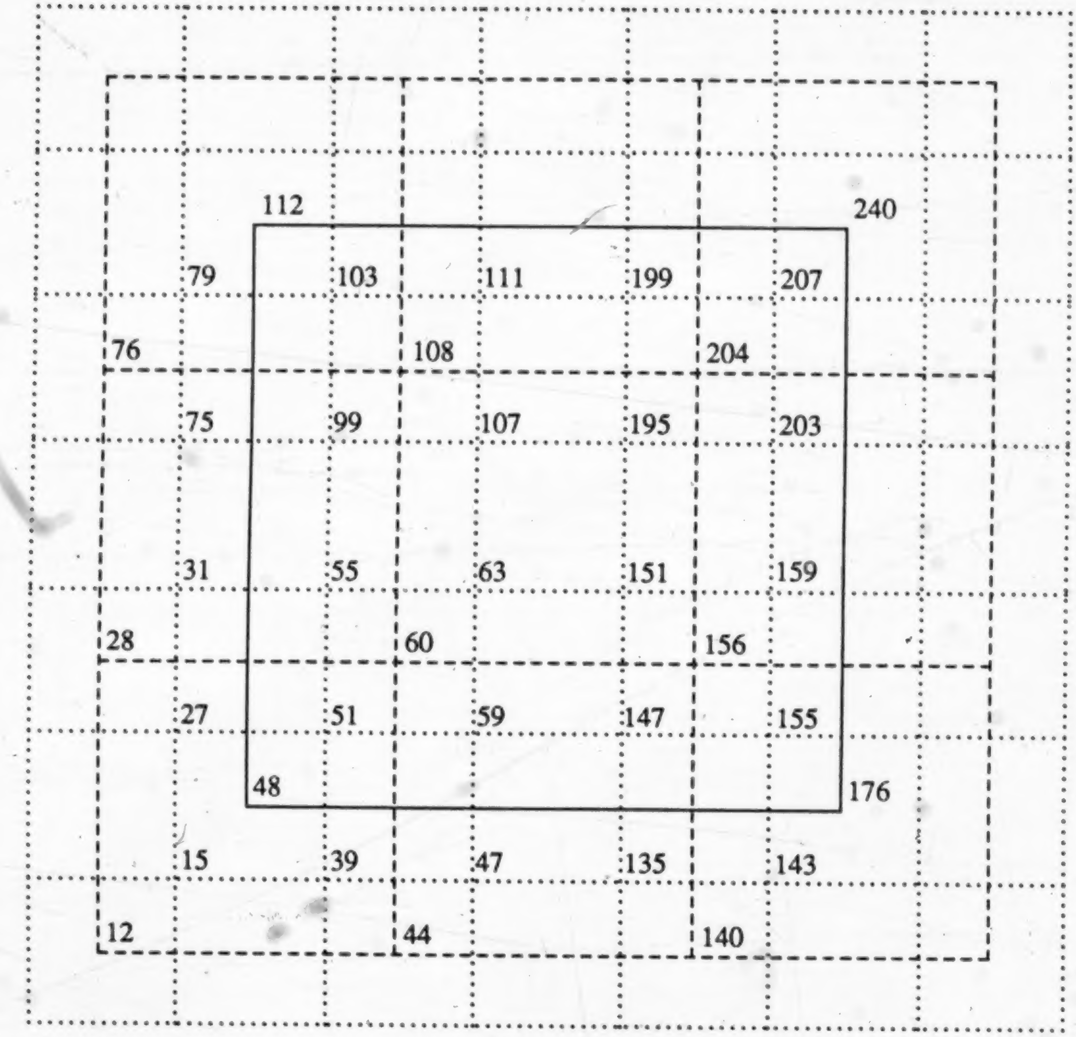

112
240
79
103
111
199
207
76
108
204
75
99
107
195
203
31
55
63
151
159
28
60
156
27
51
59
147
155
48
176
15
39
47
135
143
12
44
140

Figure 2.4 Field Tree Decomposition Principle.

immediate descendents of node 48, which covers the study area, are descendents of nodes 112, 176, and 240, which are outside the study area. Node 192, the root of the Field Tree, completes the structure making it one level bigger than a corresponding quadtree.

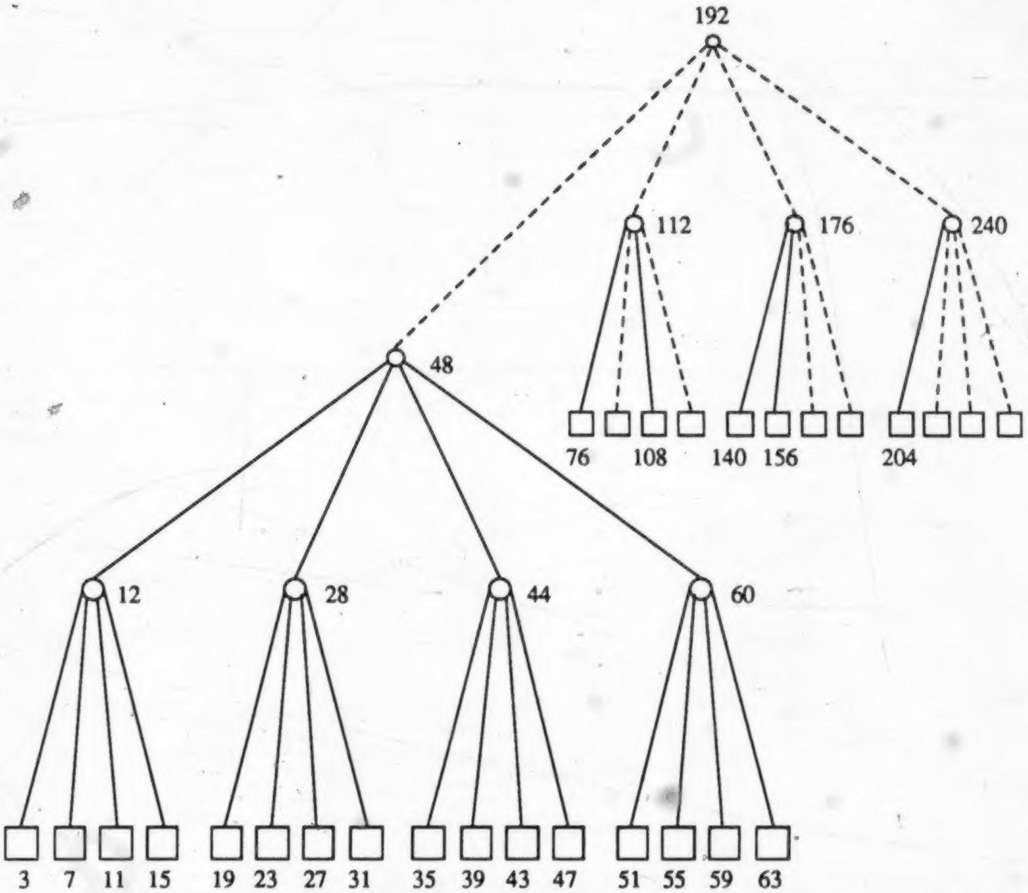

192
112
176
240
48
76
108
140
156
204
12
28
44
60
3
7
11
15
19
23
27
31
35
39
43
47
51
55
59
63

Figure 2.5 Field Tree Representation.

Not all cells of level two or deeper overlap the study area (i.e., cell 48). For example, only the center twenty five cells of level two are needed to cover the study area. The other twenty four cells need not be represented. The number of cells in a complete Field Tree of height h that overlap the study area is given by

$$1 + \sum_{i=1}^{h} (2^i+1)^2 = \frac{4^{h+1}-1}{3} + 2^{h+2} + h - 4. \tag{2.1}$$

The first term on the RHS of Equation 2.1 is the number of nodes in a complete quadtree of height h. It is evident that a complete Field Tree of height h has only slightly more nodes. In Figure 2.4, only nodes 15, 27, 39, and 51 need be stored as the descendents of node 12, but all descendents of node 60 must be stored as they all overlap the study area.

The Field Tree may be used to represent pixel-based polygon networks in a manner similar to quadtrees. Although this may not have been Frank's intent [31], using the Field Tree as a tessellation technique produces a structure comparable to the quadtree. Two characteristics of the Field Tree stand out as significantly different from a quadtree. Whereas all descendents of a homogenous cell in a quadtree are homogenous (and therefore not represented), the descendents of a homogenous cell in a Field Tree need not all be homogenous. In fact, only the center descendent is guaranteed to be of the same color (e.g., cell 60 in Figure 2.4). The other eight descendents may be multi-colored since they only partially overlap the parent cell. The second difference is the *splitting property* of the Field Tree compared to the quadtree. Consider a small polygon centered in a non-homogenous study area. In a quadtree, the polygon is split into four parts and is represented in all four quadrants of the root. In general, any region that is split by a quadtree cell boundary remains split in subsequent levels of the tree. The orderly translation of the levels of a Field Tree provides cells of diminishing sizes that cover all regions of the study area without splitting those regions throughout the tree. For example, the polygon centered in the study area is never split by cell boundaries, while a small polygon split by a cell boundary on level i is enclosed by a cell at level $i+1$ or level $i+2$.

The Field Tree has one more level than a corresponding quadtree and also requires more nodes than the quadtree for the same study area due to the shifting of levels. This implies that the spatial operations cost more with the Field Tree than with a quadtree. However, the extra level only introduces five subtrees (at nodes 76,108,140,156,204) rather than 12 subtrees as would be the case with a quadtree with one extra level. The increased storage due to level shifting is minor, as shown by Equation 2.1. The splitting property of the Field Tree makes it less effective than the quadtree for the location-based operations. For example, the point inclusion operation with a quadtree stops at a homogenous cell since there is no subtree to search. In contrast, homogenous cells in a Field Tree may have subtrees since they share descendents with neighboring cells. Since only the center descendent is known to be of the same color, only 1/9 of the time does the point inclusion operation terminate at a homogenous cell. In the remaining cases, the operation may require further search to locate the enclosing cell (in the worst case the search continues to the bottom of the tree). The performance may be improved by storing color information in the internal nodes of a Field Tree. Both the quadtree and the Field Tree require exhaustive search for attribute-based operations. Since the Field Tree is slightly larger, the cost for these operations is more expensive.

As a tessellation structure, the Field Tree compares unfavorably with the quadtree for the reasons noted above. However, there appears to be some benefit in using the Field Tree as a spatial index to geographic objects [45]. We discuss this claim in Section 3.5. In addition, the Field Tree provides a mechanism, not available with quadtrees, to enhance attribute-based search and this aspect is discussed in Section 2.6.

2.3. Pointerless Quadtrees

There has been considerable research to minimize the storage requirement and pointer overhead of quadtrees. The quadtree structures resulting from these efforts are termed *pointerless quadtrees*. They can be grouped into two classes: those that represent the quadtree as a collection of its leaf nodes and those that represent all of the nodes of a quadtree in a sequence corresponding to a traversal.

Linear quadtrees, originating with Gargantini [34], are representative of the first group of pointerless quadtrees. The original formulation was designed for the encoding of binary images and resulted in a space saving of at least 66% by the elimination of internal nodes, pointers, and WHITE leaf nodes. The remaining BLACK leaf nodes are each encoded with a *locational code* describing the location of a corner of the node and a marker indicating the size (or level) of the node. The locational code corresponds to a sequence of directional codes on the path from the root of a quadtree to the leaves. This is equivalent to taking the binary representation of the x and y coordinates of a corner pixel of a leaf and interleaving the bits from each coordinate. The leaf nodes are ordered by the locational code and stored in this sequence.

Traversing the leaf cells of a quadtree in the order of their locational codes follows a track through space known as a *Peano curve* (the locational codes are also called Peano keys). In general, this technique (also known as *Morton indexing* or Morton order [77]), is used to map n-dimensional space onto a linear representation. A significant result of this is that points close together in space will *tend* to be near to each other in the linear representation (i.e., computer memory). We call this property of the Morton order the *locality of access*. Since, in a GIS, it is often the case that the next location to be retrieved is near the current location, the Morton order tends to reduce the search effort (disk accesses) by continuing the search from the current location. Morton indexing also facilitates inorder traversal of the leaf nodes of a quadtree without any pointer overhead. The interested reader is referred to [34, 54, 62, 68, 77] for more in depth discussions of this topic.

Gargantini proposed that linear quadtrees be stored as arrays. On external storage, a simple way of doing this is by storing the nodes in a relative file with the records ordered by Morton index. Attribute-based and overlay operations are enhanced since there are fewer nodes to traverse and pointer following is eliminated. The location-based operations can take advantage of the node ordering. The point inclusion operation can be performed with $O(log\ \lambda)$ retrievals, where λ is the number of leaf nodes, by using binary search techniques [75]. The indices of nodes intersecting a rectangular window can be determined from the locational codes of two opposite corners of the window [38]. This enhances rectangular windowing in that the relevant nodes may each be retrieved by node index with binary search. Windowing with large, irregular windows, in general requires $O(\lambda)$ accesses since a linear quadtree must be traversed in parallel with a window linear quadtree to discover the overlap with the window.

Adapting linear quadtrees to represent multi-color images involves retaining all leaf nodes since each leaf node is assigned a color. This method still saves space over pointer-based quadtrees since internal nodes and pointers are eliminated. A representation for each leaf node is described by Samet, et al [75] in which each leaf is represented by a 3-tuple of <key, level, color>. The key element for each leaf is a locational code constructed from the coordinates of the southwest pixel in the leaf. The level element refers to the level of the leaf in the quadtree (with the root at level zero). Figure 2.6 shows the linear quadtree encoding for the polygon network of Figure 1.1.

< 0,2,B> < 4,2,B> < 8,3,D> < 9,3,B>
<10,3,D> <11,3,A> <12,2,A> <16,1,A>
<32,2,D> <36,3,A> <37,3,A> <38,3,D>
<39,3,C> <40,2,D> <44,2,C> <48,3,C>
<49,3,A> <50,3,C> <51,3,C> <52,2,A>
<56,3,B> <57,3,C> <58,3,C> <59,3,C>
<60,2,B>

Figure 2.6 Linear Quadtree Representation.

The linear quadtree representation of a polygon network is used to facilitate a discussion of the location-based operations in Section 2.5.

Mark and Lauzon [53] propose that the collection of leaf nodes be compressed by using a variant of run length encoding. Their technique, known as *2-dimensional run-encoding* (2DRE) discards from the linear quadtree all but the first node in any subsequence of nodes of the same color. The locational codes of intervening blocks can be recovered by knowing the codes of two successive blocks. This representation is more compact than linear quadtrees, hence the spatial operations (traversals) can be performed more rapidly.

The second class of pointerless quadtrees is obtained by a depth first traversal of the nodes of a quadtree. This formulation is called a *DF-expression* [42] and consists of a sequence of symbols: a polygon color for each leaf node and marker symbol for each internal node. For example, the polygon network of Figure 1.1 has the following DF-expression. The symbol '(' is used to represent internal nodes.

((BB(DBDAAA(D(AADCDC((CACCA(BCCCB

Figure 2.7 DF-expression.

Like the linear quadtree, the DF-expression is more compact than a pointer-based quadtree. The DF-expression is also more space efficient than the linear quadtree. Rather than storing the location and size of each leaf node explicitly as in a linear quadtree, in a DF-expression, this information is obtained by sequentially reconstructing the quadtree using the internal node symbol. Even though the nodes of the DF-expression are ordered like the linear quadtree, binary search cannot be used for the location-based operations since the sizes and locations of the nodes are implicit in the sequence. Only a sequential traversal of the DF-expression can discover this information. Thus, location operations require $O(n)$ retrievals, where n is the number of symbols in the DF-expression (which is equal to the number of nodes in the corresponding quadtree). Attribute operations also require an exhaustive traversal of the DF-expression.

2.4. Cell Methods

For small applications, the structures discussed so far are adequate, if they can be stored in internal storage. However, for large thematic applications there is usually far too much data for effective storage in memory. The direct transfer of these techniques to external storage causes problems of which the most cited is that pointer following may lead to an unacceptable number of disk accesses (i.e., retrievals) [23, 77]. This section discusses methods of representing linear quadtrees in external storage.

The B-tree [17] has been utilized for structuring the leaf nodes of quadtrees on disk [1, 75]. The B-tree, an extension of the binary search tree concept, is a dynamic structure that remains balanced under insertions and deletions and supports record retrieval in $O(\log \lambda)$ disk accesses for location-based search, where λ is the number of leaf nodes. As previously mentioned, a relative file organization of a linear quadtree offers $O(\log \lambda)$ location-based access cost. However, updating a relative file is more costly than updating a B-Tree. Although these methods are better than the $O(\lambda)$ access cost of sequential files, they are not as good as direct access or *hashing* methods which have $O(1)$ access cost. However, traditional hashing schemes are not suited to a dynamic GIS environment since a fixed amount of storage must be allocated in advance [38]. This can lead to under-utilization of storage space when it is over-estimated, or costly reorganization of files if overflow occurs due to under-estimating the space.

Several *dynamic hashing* methods have been developed that dynamically allocate storage space while attempting to maintain $O(1)$ retrieval cost. Two approaches to dynamic hashing have arisen: those that maintain a directory whose size varies as the size of the file, and those that do not use a directory. The methods which maintain a directory resolve overflow by splitting the affected bucket into two buckets, distributing the affected records across both buckets, and somehow maintaining a correspondence between the directory and the buckets. The prominent directory driven dynamic hashing method that has been applied to spatial data is *extendible hashing* [28]. This method uses a two-step process to locate a bucket: hashing determines a directory element and then a pointer within that directory element is used to locate the bucket. *Linear hashing* [49] is a dynamic hashing method that does not use a directory. Formulas are used to locate buckets directly in external storage in a one-step process. This method resolves overflow by chaining (attaching) overflow buckets as required. Bucket splitting is performed in a predictable manner so as to maintain a high storage utilization.

With both extendible hashing and linear hashing, as the amount of data increases, a new hash function must be used to address a larger key space (and vice versa as the data set decreases in size). These methods are dynamic because a hash function can be determined easily by knowing the directory size (for extendible hashing) or the number of buckets (for linear hashing). In contrast, with traditional hashing schemes the selection of a new hash function is a major consideration.

Much work has been done under the headings of *cell methods* and *bucket methods* with the aim of ensuring efficient access to point data on disk. The most successful cell methods utilize either extendible hashing or linear hashing. These methods can and have been extended to the representation of region data encoded in a linear quadtree format. Since the nodes of a linear quadtree may be ordered by locational code, cell methods may be used to organize the nodes as if they were points. The reader is referred to [23, 38] for in-depth discussions of these techniques that are briefly reviewed below.

Extendible Hashing

The locational code provides the basis of an extendible hashing function that may be used to organize the nodes of a linear quadtree in a set of fixed-size external storage areas called buckets. For 2-dimensional point data, the interleaved bits of coordinates forms a key space of binary numbers given by

$$\sum_{i=0}^{c} a_i \, 2^i , \tag{2.2}$$

where c is the length of a locational code in bits and a_i is a binary digit. An extendible hashing function h_d may be defined as

$$h_d(key) = \sum_{i=0}^{d} a_{c-d+i} \, 2^i , \tag{2.3}$$

where d is the length of the hashed (or pseudo) key and $d \le c$. The hash function h_d takes the leftmost d bits of a locational code to produce a locational code of length d. It is extendible since d may be increased to provide greater hashed key resolution when two keys are the same in their leftmost d bits.

The hash function h_d provides direct access to a directory consisting of 2^d pointers to fixed-size buckets. The first pointer points to a bucket that contains all keys for which the pseudo key consists of d consecutive zeros. This is followed by a pointer for all keys whose pseudo key begins with $d-1$ zeros followed by a 1, and so on lexicographically [28]. The last pointer is for pseudo keys consisting of d consecutive ones. To store the linear quadtree nodes of Figure 2.6 using extendible hashing, we begin with a two element directory (i.e., $d=1$) with the pointers zero and one referencing a single bucket. Each node is added to the bucket by computing its hashed key $h_1(key)$, accessing that directory element, and placing it in the referenced bucket. Assuming a bucket capacity of $b=5$ records, the insertion of the linear quadtree nodes of Figure 2.6 results in a directory and set of buckets as shown in Figure 2.8a. The hash function is h_3 and the directory has 2^3 elements.

Bucket splits are triggered by insertion of keys into full buckets. Two types of splits are possible. The first, and most common, occurs when a bucket that is pointed to (shared) by more than one directory element overflows. In this case, a new bucket is allocated, the keys are distributed across both buckets (according to the hashed key), and one directory pointer is adjusted. For example, inserting new keys 17,18,25,26, and 28 causes bucket C to overflow. A new bucket H is allocated and the resulting structure is shown in Figure 2.8b. The second type of split is triggered by an overflowing bucket that is not shared by more than one directory element. This causes the directory to double in size since a new hash function must be used. However, only one new bucket is allocated. For example, inserting a new key 13 in the structure of Figure 2.8a causes bucket D to overflow. The hash function now becomes h_4, the directory doubles in size, and a new bucket H is allocated. The resulting configuration is shown in Figure 2.8c.

The significant advantage of extendible hashing is the guaranteed $O(1)$ retrieval cost for retrieving one node of a linear quadtree. Since the bucket capacity is often greater than five, retrieving a node also tends to retrieve the neighbors of the node due to the Morton ordering of the nodes. Extendible hashing achieves a two disk access retrieval cost no matter what the size of the data set. This results in a structure that optimizes the point inclusion operation and facilitates rapid windowing operations. The price paid for this performance is the storage overhead of the directory. The directory can grow to be very large if the data is highly clustered. Each time the directory doubles in size, however, only one new bucket is allocated.

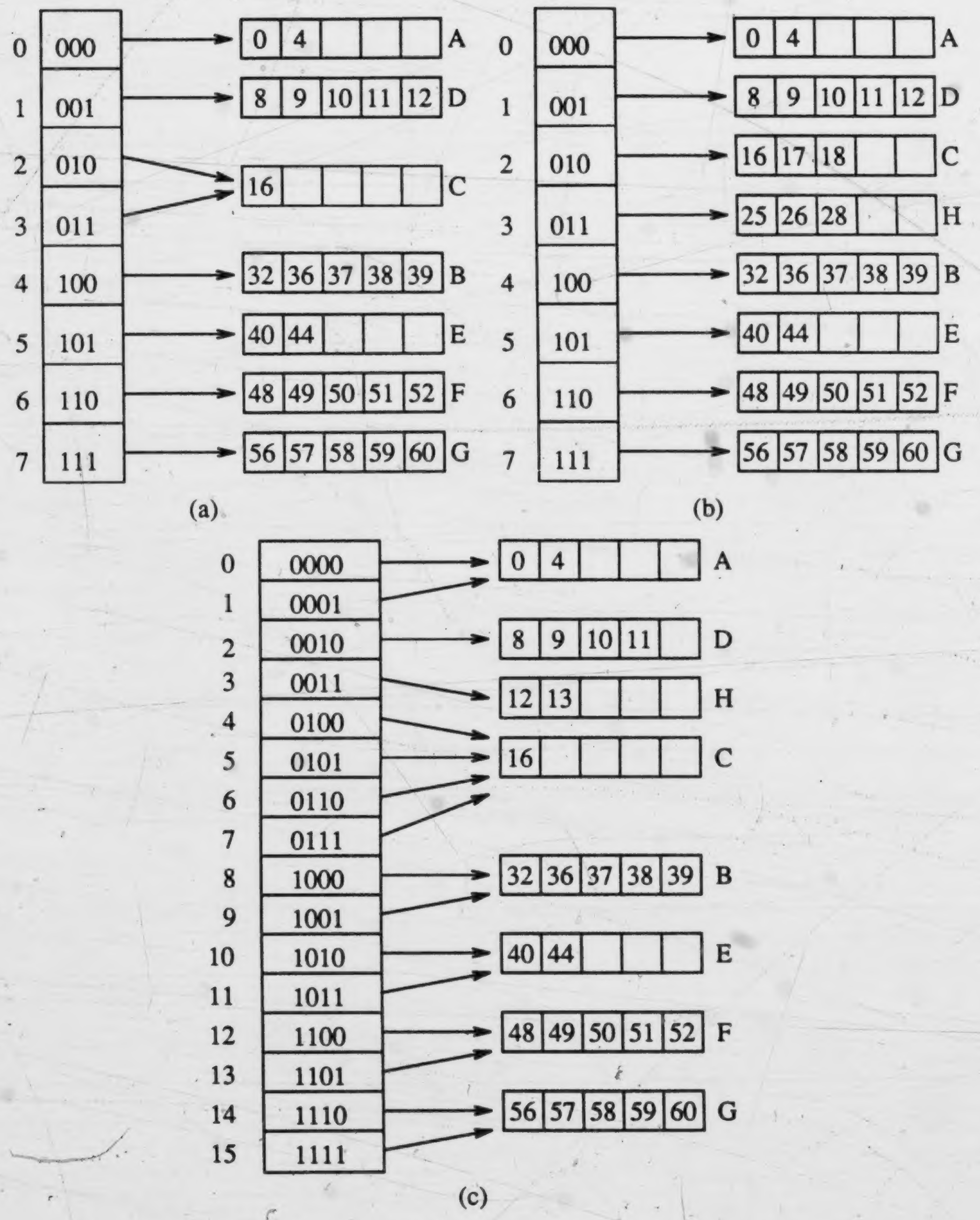

0 000
1 001
2 010
3 011
4 100
5 101
6 110
7 111
0 4 A
8 9 10 11 12 D
16 C
32 36 37 38 39 B
40 44 E
48 49 50 51 52 F
56 57 58 59 60 G
(a)
0 000
1 001
2 010
3 011
4 100
5 101
6 110
7 111
0 4 A
8 9 10 11 12 D
16 17 18 C
25 26 28 H
32 36 37 38 39 B
40 44 E
48 49 50 51 52 F
56 57 58 59 60 G
(b)
0 0000
1 0001
2 0010
3 0011
4 0100
5 0101
6 0110
7 0111
8 1000
9 1001
10 1010
11 1011
12 1100
13 1101
14 1110
15 1111
0 4 A
8 9 10 11 D
12 13 H
16 C
32 36 37 38 39 B
40 44 E
48 49 50 51 52 F
56 57 58 59 60 G
(c)

Figure 2.8 Extendible Hashing Representing a Linear Quadtree.

Since the directory of extendible hashing references the buckets by pointers, the buckets may be located anywhere on external storage. As data is added, bucket splitting causes physical scattering of the records that are logically ordered by Morton index. This implies that extendible hashing does not preserve the locality of access property of a sequential organization for linear quadtrees. Periodic assignment of the buckets to one contiguous storage area is required to physically maintain the buckets in Morton order. This assignment requires a costly reorganization of the entire structure.

The *EXCELL* method [82, 83] is an adaptation of extendible hashing to k-dimensional point data. In a 2-dimensional setting, it consists of a directory implemented as an array that provides access to a set of buckets in a manner very similar to the above technique. The EXCELL method partitions the plane into rectangular cells and the directory maps the cells onto data buckets. The decomposition induced by EXCELL corresponds to that of a region quadtree (at even levels) since all cells are split in two when the directory is doubled.

Linear Hashing

Linear hashing is a dynamic hashing method that does not use a directory to manage the cells of the hash address space (buckets). Instead, a set of hash functions similar to extendible hashing are used to map the key space directly onto a set of *bucket chains*. A bucket chain consists of a fixed-size *primary* bucket plus zero or more fixed-size *overflow* buckets attached as a linked list. The primary buckets of a set of chains are located in contiguous external storage. As records are added, overflow buckets are attached to primary buckets to resolve overflow conditions. Linear hashing is designed to maintain a high utilization of the allocated storage space using a predefined threshold. Bucket splitting is triggered by the overall storage utilization exceeding the threshold, regardless of where overflows occur. Only one bucket is split each time the storage utilization threshold is exceeded, hence the term linear hashing. The splits proceed in a predictable manner so that the hash function can always locate a bucket chain in one disk access. For 2-dimensional point data, the locational code is once again the basis of a hash function h_k which may be defined as

$$h_k(key) = \sum_{i=0}^{k} a_{c-i}\ 2^i, \tag{2.4}$$

where k is the length of the hashed key, c is defined as for extendible hashing, and $k \le c$. The hash function h_k takes the leftmost k bits of a locational code, reverses their order, and produces a locational code of length k.

Consider a set of primary buckets indexed from 0 to $2^i - 1$. The buckets are split in order of their indices so that bucket 0 is split into bucket 0 and 2^i, then bucket 1 is split into bucket 1 and $2^i + 1$, and so on until bucket $2^i - 1$ is split into buckets $2^i - 1$ and $2^{i+1} - 1$. This splitting process now repeats with the 2^{i+1} buckets, and so on triggered by too high a storage utilization. Since the index of the bucket to be split is predetermined and the index of a bucket that overflows is random, chaining of overflow buckets is used to resolve any overflow condition. Overflow keys are moved back to primary buckets when those buckets are split.

Assuming a primary bucket capacity $b = 4$, an overflow bucket capacity $b' = 2$, and a storage utilization threshold $w = 0.70$, the nodes of the linear quadtree in Figure 2.6 may be organized in external storage using linear hashing as shown in Figure 2.9. Figure 2.9a shows that seven primary buckets and five overflow buckets are required. The hash function is h_3 which may be used to access up to eight buckets. A feature of linear hashing is that at most two hash functions are required to locate the correct bucket chain for either retrieving or storing a particular record. For example, in Figure 2.9a, the hash function h_3 is used to locate any key. A query retrieving a record with a key of 58 hashes to bucket 7, since $h_3(58)=7$, which does not currently exist. In this case, h_2 is used, which hashes to bucket 3. Bucket 3 has three overflow buckets attached which means that retrieval of a record with a hashed key of 3 may require up to four disk accesses. The arrow pointing to bucket 3 in Figure 2.9a indicates the next bucket to split. If two new keys, 17 and 24, are inserted into the structure, then the storage utilization threshold is exceeded and bucket 3 is split into buckets 3 and 7 (see Figure 2.9b).

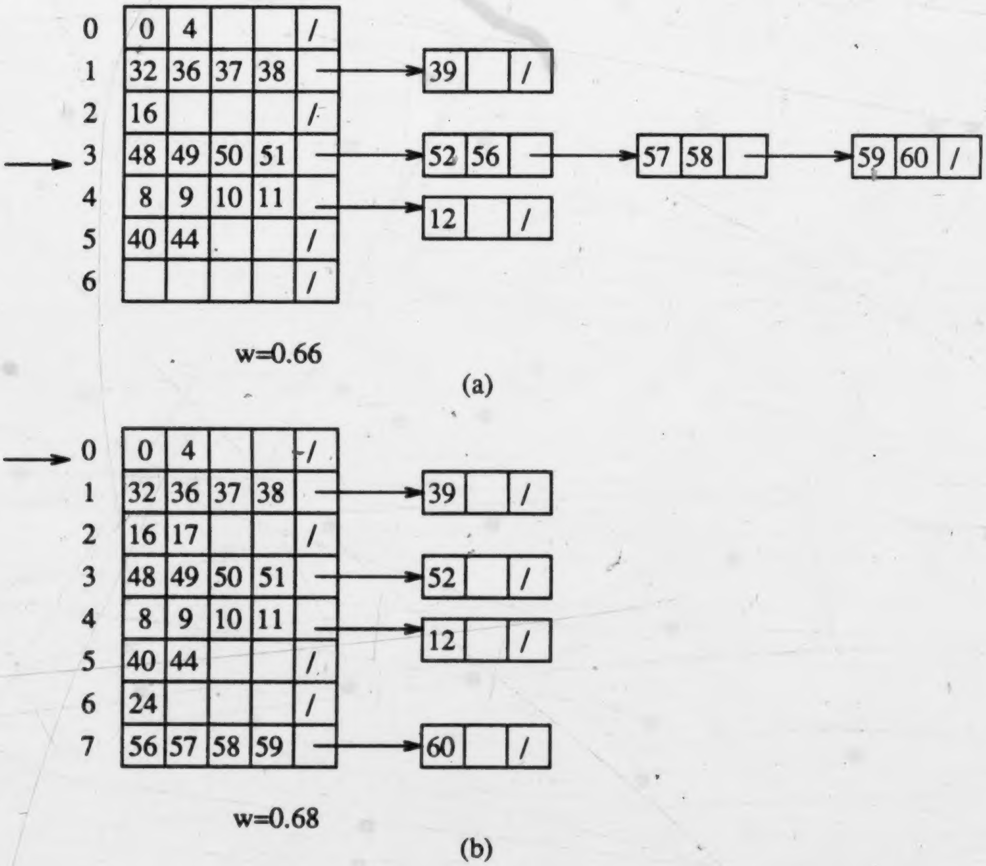

0 0 4 /
1 32 36 37 38 → 39 /
2 16 /
3 48 49 50 51 → 52 56 → 57 58 → 59 60 /
4 8 9 10 11 → 12 /
5 40 44 /
6 /
w=0.66
(a)
0 0 4 /
1 32 36 37 38 → 39 /
2 16 17 /
3 48 49 50 51 → 52 /
4 8 9 10 11 → 12 /
5 40 44 /
6 24 /
7 56 57 58 59 → 60 /
w=0.68
(b)

Figure 2.9 Linear Hashing Representing a Linear Quadtree.

Keys 56-60 can now be moved to bucket 7 since it is now accessible directly with h_3.

The next bucket to split after bucket 3 is bucket 0 since buckets 0-3 were split to create buckets 4-7. The next cycle of splitting occurs with buckets 0-7. Note, in Figure 2.9b, that the overflow chain attached to primary bucket 3 is now shorter. In general, if $b \gg 1$, then we expect a tendency for linear hashing to produce short overflow chains [49].

Interpolation-based index maintenance [9] is a multi-dimensional realization of linear hashing. It adapts the locational code to linear hashing similar to the way EXCELL adapts it to extendible hashing. The association between the cells of the search space and the buckets is obtained by a modification of the split strategy of linear hashing. When a bucket is split, the keys are distributed such that keys with smaller values remain in the original bucket while keys with larger values are moved to the new bucket (see Figure 2.9). This logically preserves the Morton order of the keys (i.e., the linear quadtree nodes). As Figure 2.9 shows however, the keys are not physically stored in this order implying that locality of access is not preserved. In fact, as the structure grows, it tends to scatter the records more and more in secondary storage.

Interpolation-based index maintenance requires overflow chains to handle collisions until the linear hashing function splits the overflowed primary buckets. However, for uniformly distributed data, short overflow chains are expected [49] implying that interpolation-based index maintenance has an expected $O(1)$ access cost for single record retrieval. This performance may not hold for clustered data. In the worst case, this method requires $O(n)$ accesses to retrieve a single node, where n is the number of nodes stored. In contrast, EXCELL guarantees $O(1)$ access cost at a price: a potentially large directory. An advantage of interpolation-based index maintenance is that storage utilization is controlled, allowing efficient use of storage space.

Hybrid Hashing

Davis and Hwang [22, 24, 39] and Hwang [38] have proposed a method that combines extendible hashing and linear hashing to produce a hybrid cell method providing a compromise between the overhead of a large directory and the advantage of high storage utilization. This method modifies linear hashing so that it uses a directory and resolves collisions by splitting rather than chaining to a predefined maximum partition depth *maxd*. Typically, this directory is maintained in memory to avoid a disk access. Extendible hashing is modified so that when the depth of the grid partition exceeds *maxd*, chaining of overflow buckets is used instead of expanding the directory. This hybrid method uses linear hashing to expand cells below *maxd*. Since a bucket associated with a cell at depth *maxd* is referenced by a directory element, the bucket may reside anywhere in secondary storage. The linear hashing technique is therefore applied in multiple areas of contiguous storage (the number of areas corresponds to the number of cells partitioned below *maxd*) rather than one area of contiguous storage [24, 39]. This effectively uses up otherwise free storage space since it cannot be predicted which cell requires expansion due to insertion of data. However, linear hashing does not have to be applied using contiguous storage for the primary buckets [49]. Litwin [49] describes methods by which the primary buckets may be stored in non-contiguous storage. In this case, a directory is used to map the hash address space to the primary buckets. While this method reduces the amount of contiguous storage required for each application of linear hashing, it does not eliminate the need for contiguous storage. Lastly, we note that this hybrid cell method also does not preserve the locality of access of linear quadtrees since linear hashing is used.

Davis and Hwang's hybrid cell method is less sensitive to the distribution of the data than the EXCELL method in that a cluster of data affects only one cell rather than the entire partition. A cluster that requires cell refinement below *maxd* is handled by linear hashing which prevents the directory portion from growing. In this regard, this hybrid cell method is similar to the *hierarchic EXCELL* method of Tamminen [84]. In hierarchic EXCELL, each cell corresponds to either a bucket or the directory of another hierarchic EXCELL file, up to a maximum depth. In this way, clusters do not cause doubling of the top-level directory size. The hybrid cell method improves upon the two disk access principle of EXCELL by providing an expected one disk access cost for record retrieval. However, in the worst case the method requires $O(n)$ accesses due to chaining of overflow buckets below *maxd*. This is particularly true if data clustering is itself hierarchical (i.e., clusters within clusters). In practice, however, thematic data sets rarely exhibit such extreme clustering, so the hybrid approach can be characterized as offering $O(1)$ location-based access with good properties: a manageable directory size, infrequent use of overflow chains, and good storage utilization.

The Grid File

The *grid file* [61] is a technique developed for the storage of multi-dimensional point data on disk. Like EXCELL, it partitions the plane into rectangular cells and uses a directory that maps the cells onto data buckets. For 2-dimensional data, the grid file uses a 2-dimensional array with each element containing a pointer to a bucket. The grid file decomposition scheme differs from EXCELL and interpolation-based index maintenance in that both the position and orientation (i.e., along which axis) of the partition lines are not fixed. Consequently, the grid file requires two additional 1-dimensional arrays (called linear scales) to keep track of the partitions along each axis. This method achieves a two disk access cost for record retrieval by a table lookup technique. Each coordinate of a point is used with one linear scale to determine and retrieve the relevant directory element. A second retrieval, using the directory pointer, obtains the data bucket.

The grid file, like the EXCELL method, is sensitive to the data distribution. Once again, overflow triggers bucket splitting and refinement of the partition. When a bucket that is shared by more than one directory element overflows, an additional bucket is allocated and one pointer is adjusted to reference the new bucket. When a bucket that is referenced by only one directory element overflows, the partition must be refined. In contrast to EXCELL, both the axis along which to split and the position of the splitting point can be chosen arbitrarily so as to resolve the overflow. This refinement of the partition splits only one cell in two thereby introducing a 1-dimensional cross section in the directory (i.e., a new row or column). With EXCELL, all cells are split in two causing a doubling of the directory. The more attractive growth of the grid file comes with the expense of maintaining the linear scales, which must be maintained in memory to ensure a maximum of two disk accesses for record retrieval. The irregular tessellation induced by the grid file makes it inappropriate for the storage of linear quadtree nodes. While linear quadtree nodes may be organized by locational code like point data, they have areal extents in standard locations and sizes induced by a regular tessellation. Using the grid file to store the nodes causes overlap conditions since the locational codes do not indicate areal extent but indicate only the splitting points. This deficiency makes the grid file unsuitable for location-based operations such as point inclusion. However, the attractive size of the grid file (compared to EXCELL) and the simplicity of its table lookup deem it a method worthy of further investigation for the representation of linear quadtrees.

All of the cell methods described above attempt to offer $O(1)$ location-based access to data buckets containing spatial data. Since these cell methods are essentially tessellation methods applied to external storage, they do not offer any improvements for attribute-based operations. While optimizing location-based operations, albeit at the expense of greater storage demands, cell methods require exhaustive search for attribute operations.

With cell methods, the organization of the data within the buckets is of minor importance since access to any part of a bucket in memory is far faster than a disk access. Thus, the buckets can store region data in the form of polygon networks in either tessellation or vector formats without affecting the retrieval performance of location-based access. If a vector format is used to organize the data contained in a bucket, then we regard this as a hybrid method (see Chapter 4). Since cell methods capture locality, the amount of processing required to perform the location-based operations can be localized to the extent and size of individual cells.

The cell methods discussed in this section have all been developed using a single storage unit model in which data is accessed serially.[1] The speed with which data may be retrieved or stored with such a model is limited by the operational speed of the storage unit. Recently, Wu and Burkhard [92] introduced a file organization scheme based on a model of simultaneously accessible multiple storage units. They present a scheme, called *M-cycle allocation*, that extends interpolation-based index maintenance to multiple random access storage units. Their analysis strongly suggests that the time T required to retrieve a set of data using a single storage unit model can be reduced to T/M using M storage units. They show that M-cycle allocation achieves near-optimum parallelism for processing of orthogonal range queries (rectangular windowing, for our purposes). The relevance of M-cycle allocation to the hybrid model developed in this thesis is discussed in Chapter 5.

---

[1] The single storage unit model may use many physical devices, but conceptually, these devices comprise one large storage unit.

2.5. Location-Based Operations

As searching techniques, tessellation structures organize the embedding space from which the data are drawn. The spatial subdivision of tessellation structures enables searching that can be localized to specific locations. This enhances location-based operations such as point inclusion. We are interested in structures that reside in external storage. Consequently, the costs of spatial operations depend on the number of disk accesses required. For a raster structure, each pixel is directly addressable and the point inclusion operation is simply a lookup operation which can be done with $O(1)$ disk accesses. Quadtrees aggregate homogenous blocks of pixels in tree structures which means that tree searches are required for location operations. In the case of pointer-based region quadtrees, this requires $O(\log n)$ retrievals where n is the number of nodes in the tree. For linear quadtrees represented with ordered relative files, a location search can be done with $O(\log \lambda)$ retrievals, where λ is the number of leaf nodes, since the leaves are in Morton sequence, the tree search reduces to searching a sorted list [75]. Using a cell method can reduce the cost of a point inclusion operation to $O(1)$ accesses.

To analyze in greater detail the cost of performing location-based operations with location-based data structures, we shall use the linear quadtree formulation. This is done primarily to facilitate the discussion of hybrid techniques in Chapter 4. First we describe the method of encoding a polygon network in a linear quadtree format. Then the costs of the point inclusion operation and the windowing operation are discussed.

The original linear quadtree formulation was designed for encoding binary images and only BLACK nodes were retained [34]. Adapting linear quadtrees to represent multi-color images simply involves retaining all leaf nodes. We use the formulation due to Samet, et al [75] which consists of a 3-tuple of <key, level, color> for each leaf (see Section 2.3).

Linear quadtrees may be stored in external storage by using methods such as an ordered relative file, a B-tree, or a cell method. The best structure to use depends on the application. For static and/or small applications, a relative file is adequate, but for dynamic applications, a tree structure is more effective for handling updates. If the latter method is used, then some of the pointer overhead eliminated by the linear quadtree formulation is reintroduced. If retrieval efficiency is paramount, then a cell method is appropriate. For the purpose of analyzing the cost of location-based operations, we choose the EXCELL representation because of its simplicity and because it offers guaranteed $O(1)$ retrieval cost. The directory consists of a direct access file of 2^d records, where d is the length, in bits, of the current hash function h_d. Each record stores a pointer to a data bucket. Each data bucket consists of a fixed amount of disk space storing a number of linear quadtree nodes.

Point Inclusion

The point inclusion operation consists of determining which linear quadtree leaf contains a given point. The input consists of the coordinates of the given point and the operation is performed by extendible hashing. A locational code is constructed from the coordinates of the given point and used to directly access a directory record. The pointer in the directory record is used to retrieve the corresponding bucket which contains a number of linear quadtree nodes in Morton order. The color of the enclosing node (polygon) is found by searching for the highest locational code that is less than or equal to the locational code of the point. Note that this search takes place in memory. Thus, the enclosing node is determined with $O(1)$ accesses.

Windowing

The general windowing operation consists of extracting a portion of a polygon network that intersects a window defined by an arbitrary polygon covering the same area of interest. We assume that both the polygon network and the window are encoded as linear quadtrees. The linear quadtree for the window is binary: BLACK nodes represent the window while WHITE nodes make up the background. Figure 2.10 shows the linear quadtree encoding of the window polygon shown in Figure 1.3.

< 0,2,W> < 4,2,W> < 8,2,W> <12,2,B>
<16,2,W> <20,2,W> <24,3,W> <25,3,W>
<26,3,B> <27,3,W> <28,2,W> <32,2,W>
<36,2,B> <40,2,W> <44,2,W> <48,2,B>
<52,2,W> <56,2,W> <60,2,W>

Figure 2.10 Linear Quadtree of a Polygon Window.

General windowing is performed by a parallel traversal of two linear quadtrees, one representing the polygon network of interest t_i and the other representing the window t_w. Those nodes or portions of nodes in t_i that overlap the nodes in the window t_w are used to construct the resulting linear quadtree t_r. The resulting linear quadtree consists of the background nodes from t_w plus the window nodes from t_w replaced by the overlapping nodes from t_i. This windowing operation entirely traverses both the window and the polygon network quadtrees. Since the extent of the window on the polygon network may be large (in the extreme, portions of all nodes of t_i may be covered), the cost of windowing is linear with respect to the size of the polygon network. That is, $O(n)$ retrievals, where n is the number of nodes in t_i. The constant for this asymptotic complexity is low since each node of both input linear quadtrees is retrieved only once.

The exhaustive traversal overhead may be avoided for *rectangular* windowing by computing the record indices of the intersected nodes directly from diagonally opposite corners of the rectangle [38]. This technique reduces the search space by identifying relevant nodes. The buckets containing these nodes may then be retrieved directly by using the locational codes of the relevant nodes. Note that each bucket is retrieved only once, even though it may contain several relevant nodes. The worst case complexity of rectangular windowing is the same as that of general windowing since the rectangle may intersect all of the nodes of t_i.

2.6. Attribute-Based Operations

Since the embedding space is organized, and not the spatial entities, attribute-based operations with tessellation models require exhaustive search. Recall that attribute-based operations take as input one or more colors and then must search for those pixels possessing the color(s). For a raster structure, an attribute search requires $O(p)$ retrievals where p is the number of pixels. In the case of pointer-based quadtrees, this search requires a complete tree traversal since all leaf nodes must be examined, that is, $O(n)$ retrievals, where n is the number of nodes in the tree. Linear quadtrees and 2DRE linear quadtrees reduce this effort since only leaf nodes are stored. The fact remains that these structures must be exhaustively searched to satisfy attribute-based operations. In this section, we discuss methods of enhancing the performance of attribute-based search and hence, the performance of the polygon set operations.

Inverted List

One way to enhance attribute searches for tessellation structures is to use an *inverted list* structure, indexed by attribute, that lists the spatial locations (pixels or blocks) for the attribute values. The inverted list can be implemented as a flat structure such as a sequential file or as a hierarchical structure as discussed in the next paragraph. This technique is equivalent to but more storage demanding than a vector representation. It is equivalent since, typically, all of the polygon boundary nodes will require a record in the inverted list. The increased storage results from having to store nodes of polygon interiors as well. In a dynamic environment, having two representations of the same data results in twice the updating overhead and costly re-inversion overhead as the data grows. In a large GIS setting, requiring twice the storage for one set of data might be unacceptable and the updating overhead would certainly be intolerable in an interactive setting.

Peuquet, et al [67, 81] have proposed a *Knowledge-Based GIS* (KBGIS) incorporating a spatial database built around a pair of hierarchical data models. The interesting aspect of KBGIS is the notion that attribute data has a hierarchical nature which can be exploited to enhance attribute-based search. The data in KBGIS is organized spatially in structures called *spatial trees* and is also organized in an object oriented approach by *object trees*. Spatial trees consist of linear quadtrees which store location and attribute data, using a separate tree for each theme. The attribute data for a theme is also organized by attribute value in object trees implemented as pointer-based *and-or trees*. Initially, the spatial trees for an application are fully defined while the object trees are initialized with only basic attribute data. The spatial trees are used to resolve both location-based and attribute-based queries. As attribute-based queries are resolved, relevant search information is retained and used to refine the object trees. Subsequent queries involving similar attributes can then be more efficiently resolved using the object trees. The effectiveness of the object trees depends on the extent of the hierarchy that the attributes of a theme admit.

Attribute Pointers

A simple way to reference the attributes in a quadtree is to use one pointer for each attribute. Each attribute pointer points to a node in the quadtree of lowest depth whose subtree completely contains the attribute. For a linear quadtree, two pointers are used to indicate a range in which the attribute may be found. In general this method reduces the search effort for attribute operations since it eliminates search for the relevant subtree or the start of a range. In the worst case, an attribute pointer(s) may reference the root of a quadtree or an entire linear quadtree. The storage overhead for this method is minimal and the maintenance of the pointers is far simpler than maintaining an inverted list.

Chen [15, 16] proposes the use of *quad tree spatial spectra* (QTSS) as a search heuristic to aid in searching the spatial trees of KBGIS. This technique involves computing statistical information regarding the spatial distribution of attribute data encoded in region quadtrees. Properties of attribute data such as density and approximate location are examples of spectra that may be computed. For each level in a quadtree, a spectrum is computed and a QTSS tree is constructed to facilitate a heuristic search procedure.

Forest of Quadtrees

Jones and Iyengar [41] introduced the concept of a forest of quadtrees, that is, a collection of the subtrees of a binary quadtree, each of which corresponds to a maximal square. Maximal squares are identified by altering the concept of an internal node to indicate something about the contents of its subtrees. An internal node is said to be of type GB if at least two of its sons are BLACK or of type GB. Otherwise the node is of type GW. Maximal squares are defined as either BLACK nodes or internal nodes of type GB. A forest is the set of maximal squares that are not contained in other maximal squares and that constitute the BLACK area of the image.

The forest concept can be extended to multi-color quadtrees by identifying the set of maximal squares for each attribute (color) in the quadtree. For the quadtree in Figure 2.3, which represents the polygon network in Figure 1.1, a forest can be constructed for each of the colors A,B,C, and D. This is done by considering one of the colors as BLACK and the other colors to be WHITE. These forests can be used as attribute directories that identify roots of subtrees that contain the appropriate color. The forest directories are obtained by storing pointers to the maximal squares for each color. Figure 2.11 shows the forest directories for the quadtree of Figure 2.3.

Color	Forest
A	6, 7, 8, f, 17, 20
B	b, 21, 25
C	13, 15, d
D	e, c

Figure 2.11 Forest of Quadtrees.

For multi-color quadtrees, this representation leads to enhanced attribute searches for a specific color since large areas of other colors can be ignored.

Field Tree

The splitting property of the Field Tree tends to reduce the partitioning of small homogenous regions (polygons) across subtrees since the levels are always shifted. The ancillary structures discussed above can be used as attribute indices to guide attribute search with quadtrees. These attribute indices reference smaller subtrees within a Field Tree than within a quadtree due to the splitting property of the Field Tree. This would seem to reduce the search effort for attribute operations, although the value of this technique requires further research to substantiate this claim.

All of the above methods involve ancillary structures that reference locations by attribute. Consequently, there is no distinction between simple and compound polygons. An attribute-based operation retrieves all regions of a particular color that, in general, is a compound polygon. To retrieve one specific region (i.e., a simple or complex polygon within a polygon network), an additional level of organization is required. Such an organization must assign a unique identifier to each region. An example of such a method consists of constructing bounding rectangles for each polygon in a polygon network. For each attribute in a theme, a set of bounding rectangle descriptions or *polygon windows* is kept. Each window is assigned a unique identifier such as a polygon number. This method permits accessing a tessellation structure by attribute or by specific polygon. This method also reduces the cost of attribute search to that of rectangular windowing with minimal storage overhead and easy maintenance of the polygon windows.

The polygon windows technique is equivalent to an attribute-based (i.e., vector) organization since the polygon object is introduced as the basis of an additional level of organization. Chapter 3 discusses attribute-based data structures in greater detail.

2.7. Thematic Overlay

The thematic overlay operation requires complete traversal of the structures representing two polygon networks. For each pair of pixels or blocks (nodes) in the input structures, the overlay operation produces an output pixel or block whose color is the union of the two input colors. For example, the overlay of two linear quadtrees representing themes is performed with a parallel traversal in a manner very similar to windowing. For any of the location-based data structures, the retrieval cost of the overlay operation is linear with respect the number of cells in the tessellations. For example, $O(p)$ with raster structures, where p is the number of pixels, $O(n)$ with quadtrees, where n is the number of nodes, and $O(\lambda)$ with linear quadtrees, where λ is the number of leaf nodes. This is the best that can be done, so those structures that reduce the amount of data stored (e.g., 2DRE linear quadtrees) are the most effective for thematic overlay. Furthermore, it should be noted that location-based data structures are much more effective for thematic overlay than methods based on attribute-based data structures, as we shall see in Chapter 3.

Chapter 3

Attribute-Based Data Structures

3.1. Introduction

The class of attribute-based data structures is analogous to the traditional map wherein thematic content is controlled and the location of polygon boundaries varies according to changes in attribute value. The spatial distribution of attributes defines the polygons in a polygon network and the attribute values are associated with the polygons (i.e., locations are associated with attributes). This contrasts with location-based data structures which associate attribute information to locations imposed by a tessellation of the plane. The distinguishing characteristic of attribute-based data structures is that polygons may be ordered by attribute to permit rapid execution of attribute-based operations.

Common to all attribute-based data structures is the representation of points by 2-d coordinates. Curves are represented by sequences of points (vectors) and regions are constructed by considering closed sequences of vectors. Due to this formulation, this class is also known as the class of vector data structures. Before discussing vector data structures, we define some of the terminology that is used. A 2-d coordinate pair defines a *point* or *vertex*. The straight line segment between any two vertices is called an *edge*. Vertices that are common to three or more edges (two or more polygons) are called *nodes*. The sequence of vertices (set of edges) between any two nodes is a *chain*. For this thesis, we designate a *curve* to be either an edge or a chain. The size of a vector representation of a polygon network is $O(e)$, where e is the total number of edges in the network. Figure 3.1 shows a vector representation of the polygon network of Figure 1.1.

The coordinates defining vertices are the most storage demanding aspect of vector

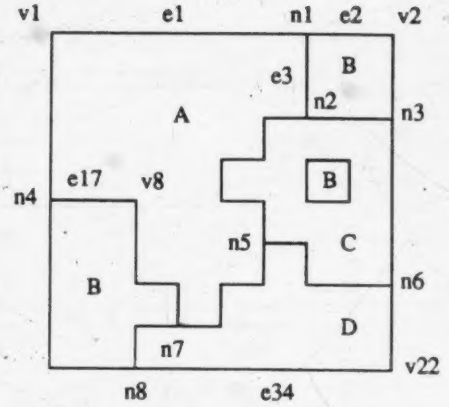


Figure 3.1 Vector Representation of a Polygon Network.

structures. The term *chaincode* refers to a well known family of techniques used to reduce the space required for coordinates. This method generally consists of defining chains in terms of unique directional codes that represent unit length vectors. The original formulation of Freeman [33] defines eight symbols for four unit length vectors in the principal compass directions and for four $\sqrt{2}$ length vectors in the four diagonal directions (see Figure 3.2).

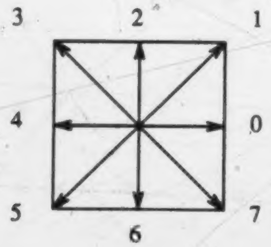


Figure 3.2 Eight Direction Chaincodes.

A chain or polygon boundary is approximated by specifying the coordinates for the starting location and a sequence of chaincodes. With eight directions, each code requires three bits for storage. The precision of the chaincode approximation may be altered by defining different sets of chaincodes [68]. A chain or polygon boundary may be equivalently represented by either chaincodes of sufficient precision or a sequence of 2-d coordinates. The significant difference between the two representations is the storage reduction that may be achieved by using chaincodes. For thematic data composed of highly irregular chains (not straight lines), the use of chaincodes requires considerably less storage than sets of 2-d coordinates.

There are two general methods with which to organize thematic vector data. *Polygon-driven* data structures use the polygon object as the basic unit and list the edges or chains for each polygon. *Curve-driven* structures use the edge or chain as the organizational object and list the polygon information for each edge or chain. Our discussion of vector data structures is guided by this classification but it should be noted that some of the methods presented use both organizational techniques.

3.2. Polygon-Driven Structures

The most primitive polygon-driven structure is the *spaghetti* model which is a direct one-to-one encoding of the boundaries of the polygons on a map [20]. Each polygon is represented in a polygon table by listing the edges (vertices) that make up its boundary. This results in the storage of shared edges twice and can cause gap and sliver problems if the edges are doubly digitized.

An improvement of the spaghetti model is the *point dictionary* [65] in which the coordinates of every vertex in a polygon network are stored only once in a table. For each polygon, a sequence of pointers to records in a chain file indicates the chains making up the boundary. For each chain, the chain file lists the vertices (or chaincodes) comprising it. This method eliminates the redundant storage of points and avoids the sliver and gap problems of the spaghetti model. An example of such a structure is given by Weiler [91] and is described in detail in Section 3.4.

A number of vector data models have also been proposed by investigators of geometric complexity. These are elaborate models for representing polygons and are characterized by the use of *hierarchical* organizations such as rectangular approximations of polygon boundaries. We briefly review two of these methods.

The *Binary Searchable Polygonal Representation* (BSPR) [11] is a bottom-up approach to curve approximation using a hierarchical set of bounding rectangles. In this method, a curve (e.g., the boundary of a polygon consisting of a chain) is decomposed into *simple sections* that are monotonic in both the x and y directions. Upright rectangles are used to approximate simple sections and a tree structure is built by repeatedly combining adjacent simple sections into compound sections until the entire curve is covered by one compound section. In [11], Burton shows how the BSPR can be used for performing the point-in-polygon operation and the polygon intersection operation with algorithms based on binary search. Note that the resolution of the BSPR approximation is fixed, that is, once the width of the rectangles for simple sections is chosen, it cannot be varied.

Strip trees [5] are a similar method for the representation of curves. This top-down approach starts with a rectangle enclosing the entire curve. This rectangle is recursively decomposed into two parts or strips at a point common to both the curve and the rectangle. This splitting recurs until all strips are of widths less than some threshold. The leaf nodes of a strip tree approximate a polygon boundary with arbitrary precision. In the limit, the strips correspond to the edges of a polygon and the strip tree is a binary tree organization of the polygon vertices. The strip tree is an improvement over the BSPR as an approximation device in that strip tree rectangles have arbitrary orientations while the rectangles of BSPR are all upright. Furthermore, the resolution of the strip tree is not constrained by an a priori choice of rectangle width as is the case with the BSPR.

Each of the above data structures provides a mechanism for representing one polygon in a polygon network. Implicit in all of these organizations is the existence of a *polygon table* (which is why they are called polygon-driven). For each polyon in the network, the polygon table provides a pointer to the data describing its boundary and, for the hierarchical structures, also its interior. Complex and compound polygons in the network can be accommodated by introducing some additional organization of the polygon table. A mechanism, due to Weiler [91], for hierarchically organizing complex and compound polygons is described in Section 3.4.

Using polygon-driven structures, attribute-based operations are cost effective. Given an attribute, it is necessary first to consult the polygon table to determine the relevant polygons. The pointers associated with the polygons provide access to only that (boundary) data associated with the relevant polygons. Since the polygon table may be ordered or indexed by attribute, the attribute-based operations may be accomplished with $O(e)$ retrievals, where e is the number of edges comprising the polygons with the desired attribute.

Generally, however, there is no spatial ordering of the polygons in the polygon table (but see Section 3.5 where we discuss a technique that achieves some success in this regard). Consequently, location-based operations require the exhaustive retrieval and processing of polygons in the network. Both the point inclusion operation and the windowing operation require $O(e)$ retrievals, where e is the number of edges comprising the polygons, since each polygon must be tested in the worst case. The thematic overlay operation for two polygon-driven structures requires $O(e^2)$ retrievals since the determination of the intersection points of the set of e edges is an $O(e^2)$ operation.

Despite these superficial similarities in the performance obtained with all of the polygon-driven structures, there are some significant differences. The hierarchical models provide improved asymptotic cost performance of most polygon operations *once* all of the relevant polygons have been identified and retrieved. For example, using the BSPR, the processing costs of the point inclusion operation and the polygon intersection operation are $O(\log e)$ and $O(e \log e)$ operations, respectively, where e is the total number of relevant edges. In contrast, these operations have cost complexities of $O(e)$ and $O(e^2)$ operations, respectively, when using the spaghetti or point dictionary structures.

The hierarchical models enable performance of the spatial operations as tree searches, but this performance does not apply to polygon networks. For example, the BSPR does not reduce the search effort required to identify candidate polygons for the point inclusion and set operations, or to determine the overlap of a window with a polygon network. In fact, the storage required by the BSPR is typically about twice that required for the vertices alone [11], making it more storage demanding than the point dictionary yet not offering performance improvements. Similar comments may be applied equally to the strip tree. For these structures, the decomposition criterion is governed by the data. Hence they require an enormous amount of updating overhead whenever the polygons change - the entire structure must generally be reconstructed from scratch. While these structures may be viable for a static spatial database application, they do not appear to be desirable in the case of a dynamic GIS setting.

In summary, we regard the point dictionary as the most desirable polygon-driven vector data structure.

3.3. Curve-Driven Structures

The basic curve-driven data structure involves the use of a list of edges to organize vertex and polygon information. In its most primitive form, the *edge file* is the only structure used and, for each edge, lists the start vertex, the end vertex, the left polygon, and the right polygon. The information provided by the edge file is sufficient to perform all of the usual spatial operations. Point inclusion is resolved by first searching for that edge which is closest to the point. Then determining which side of the edge the point is on, gives the enclosing polygon. Windowing is accomplished by intersecting all edges with the edges of a window polygon. Attribute retrieval consists of finding all edges having the attribute (polygon) on their left or right side. All operations, with the exception of edge retrieval, using this primitive data structure, require exhaustive searching of the edge file. A simple example of this structure is discussed by Dangermond [20]. The Dual Independent Map Encoding (DIME) model [87] is a well known realization of this approach.

The curve-driven data structure has one significant advantage over polygon-driven methods. Once the boundaries of a polygon have been retrieved, information about neighboring polygons is immediately available (since left and right polygons are given for each edge). Because of the availability of this neighborhood information, the curve-driven data model has been termed a *topological model* by some authors (e.g., [68]). Figure 3.3 shows the basic curve-driven data structure for the polygon network of Figure 3.1. Figure 3.3a shows a portion of the edge file and Figure 3.3b depicts a portion of a point dictionary. Note that the node and vertex elements in the edge file are pointers to the point dictionary so as to avoid duplication of coordinates.

The curve-driven data structures that organize polygon information using edges are best suited for applications where the number of edges per polygon is small, or where

Edge	Left Polygon	Right Polygon	Start Vertex	End Vertex
e1	/	A	v1	n1
e2	B	/	v2	n1
e3	B	A	n1	n2
e34	D	/	n8	v22
e17	A	B	n4	v8
⋮	⋮	⋮	⋮	⋮

(a)

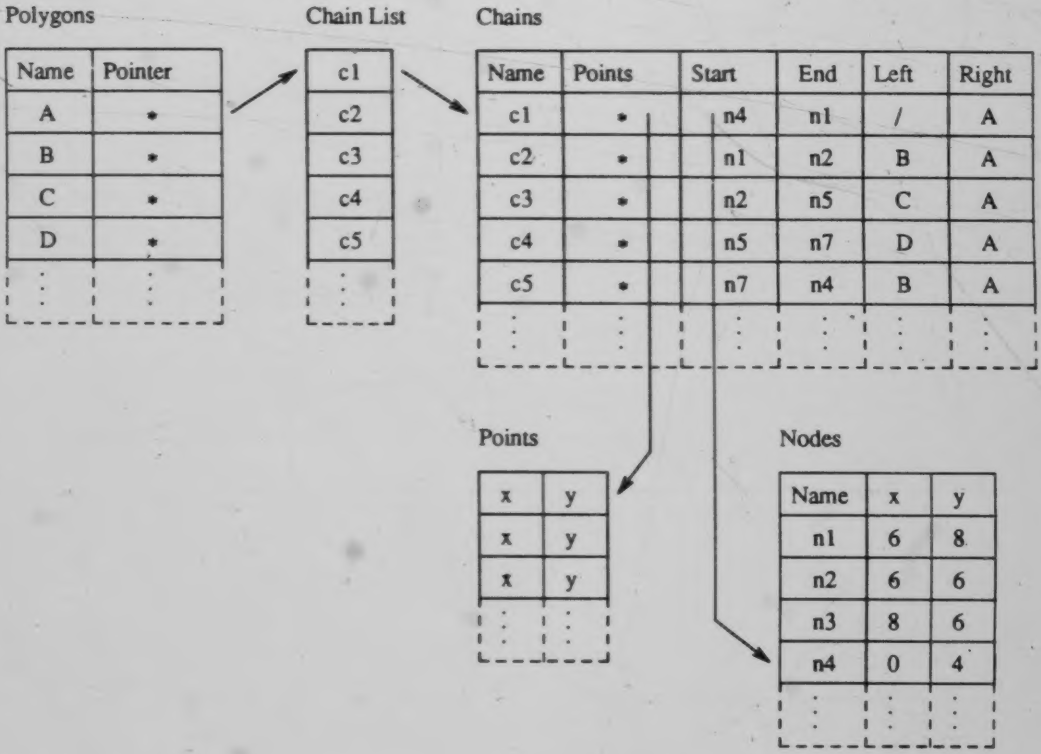
Vertex	x coordinate	y coordinate
v1	0	8
n1	6	8
v2	8	8
v22	8	0
n8	2	0
v8	2	4
⋮	⋮	⋮

(b)

Figure 3.3 Curve-Driven Data Structure.

linear features are the central aspect. The DIME model, for example, was originally designed for storing street maps to aid in data capture for the U.S. Census. In a DIME file, edges are used to define linear features such as streets, blocks, railroads, and rivers.

For thematic data, where the number of edges comprising a polygon is large, a curve-driven data structure using a chain, rather than an edge, as the basic element, is more appropriate. The POLYVRT data structure [65] is one example of such an approach. POLYVRT overcomes the serious retrieval inefficiencies of the previous methods by introducing a hierarchical organization for the separate storage of vertex, chain, and polygon information (see Figure 3.4).



Polygons

Name	Pointer
A	*
B	*
C	*
D	*
⋮	⋮

Chain List

c1
c2
c3
c4
c5
⋮

Chains

Name	Points	Start	End	Left	Right
c1	*	n4	n1	/	A
c2	*	n1	n2	B	A
c3	*	n2	n5	C	A
c4	*	n5	n7	D	A
c5	*	n7	n4	B	A
⋮	⋮	⋮	⋮	⋮	⋮

Points

x	y
x	y
x	y
⋮	⋮

Nodes

Name	x	y
n1	6	8
n2	6	6
n3	8	6
n4	0	4
⋮	⋮	⋮

Figure 3.4 POLYVRT Data Structure.

Rather than an edge file, it maintains a chain file where, for each chain, left and right polygon identifiers and the start and end node pointers are stored. The coordinates of nodes are stored in a separate node dictionary. The coordinates of points between nodes are kept in a separate point file. For each chain, a pointer references the beginning of the appropriate set of coordinates.

In addition to the chain file, POLYVRT also maintains separate lists that assemble the boundary chains for each polygon. A polygon table is used to store a pointer for each polygon that references the appropriate chain list. The dual organization by polygon and by chain allows access by polygon identifier and by chain identifier. This additional structure makes POLYVRT a polygon-driven method as well.

The distinction between points and nodes in POLYVRT is an important one. It has no effect on polygon adjacency relationships and leads to processing and storage advantages [65, 68]. Queries concerning the adjacency of polygons need only consider the polygon and chain information. This makes these types of operations dependent on the number of polygon chains rather than the number of points making up polygons (as with DIME). In terms of retrieval costs, the coordinates of non-node vertices need only be retrieved when required.

Since POLYVRT incorporates a polygon table, attribute-based operations do not require exhaustive search. The table may be ordered or indexed by attribute as with the polygon-driven methods. For the location-based operations, POLYVRT offers no retrieval improvements since potentially every polygon must be tested. The hierarchical organization of POLYVRT makes it a highly pointer-based structure and the search required for location-based operations can lead to an unacceptable number of page faults.

An interesting topological vector method has been designed by the U.S. Bureau of the Census for its Topologically Integrated Geographic Encoding and Referencing (TIGER) system [56]. TIGER will automate the mapping and related geographic activities required for the Bureau's survey and census programs, starting with the 1990 Decennial Census.

The core structures of TIGER are randomly sequenced lists, one for each of three topological elements: 0-cells, 1-cells, and 2-cells which correspond to nodes, chains, and polygons respectively. The lists are maintained as separate files and contain both locational data and attribute data. TIGER provides for the linkage of attribute information to each element in all three lists by way of pointers to auxiliary structures.

The central elements of TIGER are chains which are stored randomly in a chain file. The TIGER chain file is similar to the POLYVRT chain file in that it stores, for each chain, the start/end nodes and the left/right polygons. Each chain record points to another chain record having a common node as an endpoint. This technique is referred to as *threading* and is used throughout the TIGER structure. This type of threading can be used to determine the chains around each node. In addition, the chain file threads the chains around polygons. Like POLYVRT, the coordinates of points are stored separately in a point file and are referenced by pointers in the chain file.

TIGER also incorporates an additional structure, a polygon table, which has one record for each polygon in a polygon network. Each record stores a pointer to the first chain in the chain file which has the polygon on its left or right side. Only one such pointer is needed since the other chains bounding the polygon are threaded from the first chain record. A polygon directory is used to index the polygon table which allows for the ordering of polygons by attribute. This makes for rapid attribute retrieval.

In addition to the polygon directory, access to polygons with TIGER may be achieved by other directories which index *geographic covers*. A geographic cover combines groups of polygons into geographic areas. Examples of covers are boundaries of states, counties, and census tracts. Geographic covers are indexed by directories that may be sorted according to access needs [56]. For each cover, a pointer references the first polygon in the polygon table that is contained in the cover. The other polygons making up the cover are threaded within the polygon table. The use of covers enhances attribute operations by predefining often accessed sets of polygons.

The TIGER model is similar to the POLYVRT model in the use of a vertex, edge, polygon hierarchy. Both models are highly pointer-based in that POLYVRT uses lists of pointers to files while TIGER threads sets of pointers within files. In terms of attribute-based operations, similar comments apply to TIGER as to POLYVRT. TIGER is, however, a significant development in terms of the location-based operations. This is discussed in greater detail in Section 3.5.

There are many other examples of vector data structures similar in concept to the so-called topological model. Some of these are used as the underlying structures of commercial GIS's. These are discussed in Section 4.4.

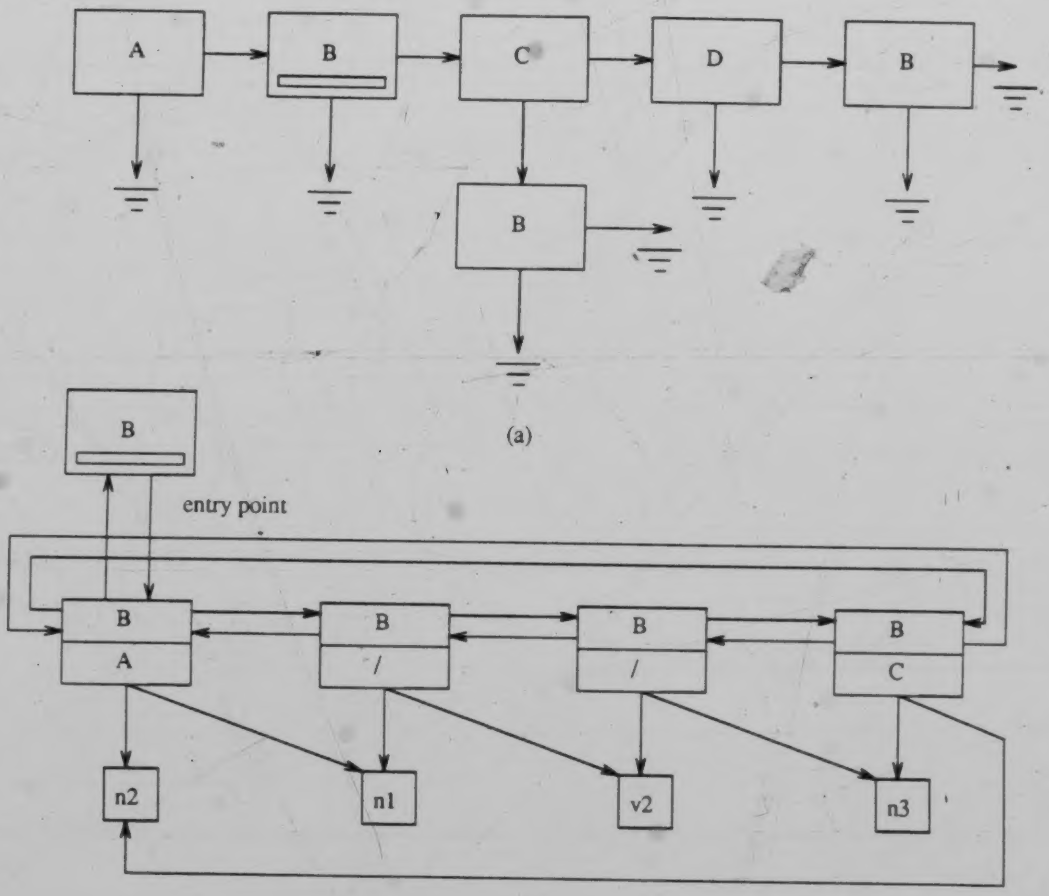
3.4. Attribute-Based Operations

Since vector models define spatial objects, the objects can be organized to permit rapid retrieval based on attribute. For polygon networks, a polygon-driven organization accomplishes this by providing an inverted list of polygons indexed by attribute. Vector data takes up the same of amount storage whether it is stored in an inverted list or not. The extra storage required for an index to the inverted list is minimal since only one pointer for each attribute value is required. This technique essentially orders a polygon table by attribute. Each pointer references the start of a group of polygons possessing the relevant attribute. This permits a query for a particular attribute to retrieve the polygon data with $O(e)$ accesses, where e is the number of edges possessing that attribute. Additional organization of the polygon table can be used to facilitate retrieval of individual regions in a polygon network (simple or complex polygons). As described for location-based data structures, each region must be assigned a unique identifier. The holes in complex polygons must be identified to allow correct resolution of attribute-based operations.

To analyze in greater detail the cost of performing attribute-based operations with attribute-based data structures, we shall use the structure proposed by Weiler [91], with some modifications. Once again, as in Section 2.5, this is done primarily to facilitate the discussion of hybrid techniques in Chapter 4. First we present Weiler's data structure and then we discuss the costs of the polygon set operations.

Weiler introduces a two level data structure to represent a collection of spatial objects called *contours* that, for our purposes, are equivalent to polygons. The top level of the structure is a binary tree that models two geometric relationships between polygons: coexistence and containment. Two polygons coexist if neither polygon lies within the area of the other (i.e. non-overlapping), while containment implies that one polygon lies within the area of the other. For a polygon network, the definition of coexistence is relaxed to include adjacent polygons whose overlap consists of common edges. Each node in the binary tree represents one polygon. One branch points to a polygon that coexists within the same area while the second branch points to a polygon that is contained within the area of the parent polygon. This method is general enough to accommodate arbitrarily complex polygons in a polygon network. This is a significant improvement over the other vector structures.

Each node in the binary tree contains an entry point to a *graph structure* which is a vector encoding of the polygons. The graph structure consists of polygon, edge, and vertex elements that are linked together to represent adjacency relationships. The entry point references the start of a circular list of the edges forming the boundary of a polygon. The list is doubly linked to allow clockwise and counter-clockwise processing of the edges. Each edge points to the vertices that define it, while the entry point edge also points back to the parent polygon node. Figure 3.5 shows Weiler's data structure for the polygon network of Figure 3.1. Figure 3.5a shows the binary tree structure and Figure 3.5b depicts a portion of the graph structure.


A
B
C
D
B
B
B
entry point
B
A
B
/
B
/
B
C
n2
n1
v2
n3

(a)

(b)

Figure 3.5 Weiler's Data Structure.[2]

For the purposes of Chapter 4, we introduce a modification to Weiler's data structure. A study area may correspond to a conventional cartographic map. We refer to polygons that are completely within the map as *closed* while those polygons that cross the map boundary are termed *open*. The boundary of the map may be viewed as a set of pseudo-edges that complete open polygons. The modification to Weiler's data structure is made within the graph structure and consists of identifying pseudo-edges. See [91] for details on a realization of the data structure.

---

[2] Modified from Weiler [91].

Polygon Set Operations

In general, the polygon set operations involve two polygon networks n_1 and n_2 and a polygon from each network p_1 and p_2. The intersection operation obtains a polygon p_3 such that $p_3 = p_1 \cap p_2$. The union operation obtains a polygon p_3 such that $p_3 = p_1 \cup p_2$. The difference operation obtains a polygon p_3 in either of two ways: $p_3 = p_1 - p_2$ or $p_3 = p_2 - p_1$.

Weiler introduces a *polygon comparison* algorithm that can generate, in a single application, all of the required information to perform the polygon set operations. The comparison process resolves two basic spatial relationships between the boundaries of the input polygons: intersection relationships and enclosing relationships. The algorithm takes as input two or more graph structures representing the input polygons and produces a graph structure for the output polygons. Each output polygon is characterized as to its boundary, holes, and *ownership*. Ownership is a function reflecting all of the input polygons that own or overlap the area of an output polygon. In essence, the comparison algorithm performs an overlay of the two input graph structures and determines the ownership of each output polygon. The polygon set operations are solved by a selection of one or more groups of output polygons.

The polygon comparison algorithm consists of four steps:

1. Merge: The input polygons are merged into a single graph structure by finding the intersections between edges. All output polygons are embedded in the output graph.

2. Traversal: Each polygon boundary (contour) is traversed to discover its ownership.
3. Disjoint contours: Additional tests are made to characterize any disjoint boundaries or any disjoint graphs of polygons. One result of this is that holes are associated with their containing polygons.
4. Selection: The output polygons can be grouped by ownership. The results for any of the polygon set operations can be determined by selecting the appropriate group(s).

See [91] for a detailed description of the polygon comparison algorithm.

The simplest approach to performing the polygon set operations is to use Weiler's comparison algorithm to compare two polygon networks. The result is then obtained for the desired operation by selecting the output polygons on the basis of ownership. Using the above notation, if the operation is

intersection	then select polygons owned by p_1 and p_2.
union	then select polygons owned by p_1 or p_2.
difference	then select polygons owned by p_1 and not by p_2, or select polygons owned by p_2 and not by p_1.

Weiler's structure is a general approach to representing polygon networks with arbitrarily complex polygons. For polygon networks consisting of only simple polygons, this structure reduces to a sequential list of coexisting polygons. In this case, the cost of attribute retrieval is the same as that of the other vector structures. Searching of more complex polygon networks is more costly since holes must be searched. In this case, coexisting polygons at each level of the binary tree in Weiler's structure can also be ordered by attribute. This permits faster retrieval of the relevant polygons while still providing the general solution of the graph comparison algorithm.

3.5. Location-Based Operations

From the standpoint of searching techniques, vector models are characterized by structures that organize the data to be searched (i.e., the edges comprising polygons)—not the embedding space. Location-based search through vector models must proceed by testing each polygon for inclusion of a point or for intersection with a window. In the worst case this may mean exhaustive search: $O(e)$ retrievals, where e is the number of edges. This is easily seen if we consider that the vector structures with polygon tables order those tables by attribute to optimize attribute-based operations. It seems reasonable to introduce another index to the polygons, based on location, to enhance location-based search. An example of this approach is described in [8]. Unfortunately, it is not generally possible to transform 2-dimensional space into a linear representation and preserve all spatial properties of spatial data (e.g., neighborhood or nearness). The locational code transformation gives a linear representation that admits a total ordering, but it does not correspond to any ordering in two dimensions. However, there are methods for enhancing the performance of location-based retrieval and hence, the performance of the point inclusion and windowing operations. In this section, we describe three such methods.

Polygon Table Organization

A simple and common method of associating location information with the polygons in a polygon table is by describing enclosing figures for each polygon and inserting them into the polygon table. The enclosing (or bounding) figures are simple polygons, such as rectangles and circles, which are easy to process. Location-based operations are enhanced since the polygon table may be searched by testing the enclosing figures without retrieving the polygons.[3] In this way, both potential polygons and irrelevant polygons may be determined quickly. This technique avoids the retrieval of irrelevant polygons for testing. However, this technique still entails sequential traversal of the polygon table. To reduce search through the polygon table, some other organization is required.

---

[3] TIGER also stores in each chain record a bounding rectangle description for the chain.

The binary tree organization of polygons by Weiler's structure corresponds to a polygon table. It can enhance the point inclusion operation by using a tree search approach. Irrelevant polygons may be avoided by following coexisting polygon branches when the current node does not contain the point and following contained polygon branches when the current node contains the point. However, in the worst case, this method requires exhaustive traversal of the tree. A similar approach can be used for the windowing operation to avoid testing contained polygons when the enclosing polygon does not intersect the window.

Guttman [35] describes a dynamic structure called an *R-Tree* that can be used to index n-dimensional objects stored in a spatial database. R-Trees can be used to index the polygons in a polygon network. The basis for the R-Tree is the hierarchical organization of the bounding rectangles of polygons. An R-Tree is similar to a B-Tree with each node containing records with two fields: a bounding rectangle and a pointer. Each leaf node contains *index records* in which the rectangle bounds a simple polygon and the pointer references the vector data describing the polygon. For each record in an internal node, the pointer indicates a lower node in the R-Tree and the rectangle bounds all the rectangles in the records of the lower node. Building an R-Tree requires computing the bounding rectangle of each polygon and inserting the index record of the polygon into the tree. The shape of an R-Tree is governed by the order in which the index records are inserted. Figure 3.6 shows the R-Tree realization for the polygon network of Figure 3.1. Figure 3.6a shows the R-Tree index and Figure 3.6b depicts the bounding rectangle hierarchy. The rectangles were inserted in the order R19,R20,R18,R17,R16,R15 using Guttman's algorithm.

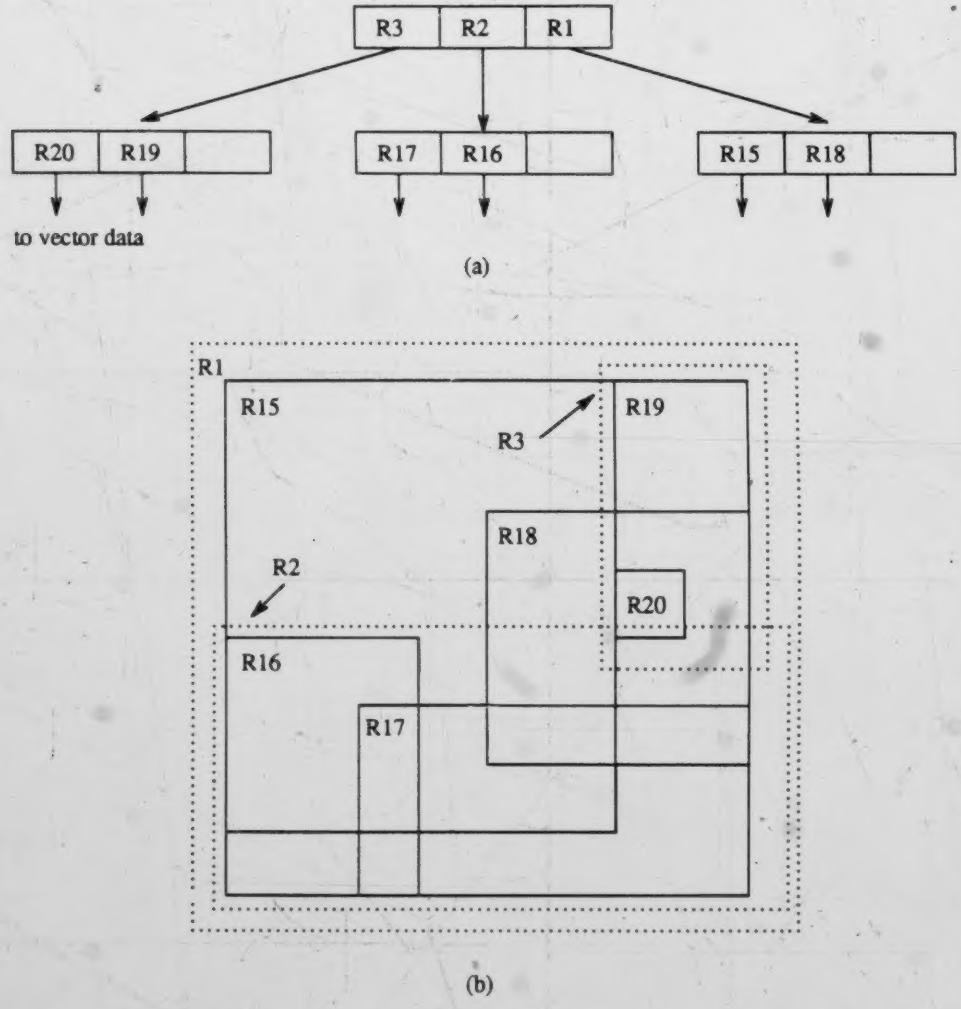

R3
R2
R1
R20
R19
R17
R16
R15
R18
to vector data
(a)
R1
R15
R19
R3
R18
R2
R20
R16
R17
(b)

Figure 3.6 R-Tree Data Structure.

The R-Tree introduces a hierarchical index to the polygons that permits location-based searches that can be performed with a logarithmic number of retrievals, based on the fanout of the tree. Each node in an R-Tree has between *m* and *M* records which means that the height of an R-Tree containing n polygons is at most $\lceil \log_m n \rceil - 1$. This represents an improvement over the methods described in the previous paragraphs. Unfortunately, for a polygon network, the set of bounding rectangles may have an excessive amount of overlap (see Figure 3.6). This implies that more than one subtree of a node may need to be searched and tested for the point inclusion operation. The inclusion testing requires the retrieval of polygon data. Thus, the cost mentioned above is only a best case value for a retrieval cost that may, in the worst case, require exhaustive traversal. Variations of the R-Tree that reduce or eliminate the overlap problem are described in [73] and [29], respectively.

For the windowing operation, the R-Tree provides a mechanism for performing a fast rectangle check for real or potential intersections between the polygons and the window. Only those polygons which do or potentially intersect the window need be retrieved for this operation. If the R-Tree index is maintained in internal storage, this method requires the least amount of disk access for windowing.

Cell Methods

A polygon network may be organized with a regular tessellation by partitioning the network into cells. Each cell either references those polygons that are contained within it or intersect it, or each cell contains the portion of the polygon network that overlaps with it. The former method is once again a polygon table organization but the table entries are induced by a regular tessellation. The latter technique is similar to methods discussed in Chapter 2. Two variations are described below.

Techniques similar to cell methods can be used to organize vector data. For example, Tamminen [82] describes an edge variant of the EXCELL method that partitions a polygon network into cells and stores polygon information in the buckets. This *edge-*

EXCELL method can be used to localize location-based search to only those cells that are relevant. The vector data is retrieved from the buckets attached to the relevant cells and a location-based operation is applied to this data.

The Field Tree [31] may be viewed as a cell method that attempts to store a polygon in the smallest cell which completely encloses it. The systematic shifting of cells is designed to vary the locations of the splitting lines around the plane so that cells may be located which do not partition polygons. The intent is to store relatively small polygons deeper in the Field Tree thus avoiding the overflow of buckets attached to higher level cells. This method is unsuitable when there are many relatively large polygons since large polygons crossing cells boundaries cannot be avoided.

Applying the above cell methods to vector models seems to offer location-based searches at costs approaching those of tessellation methods. These approaches can be viewed as combining both tessellation and vector techniques in hybrid fashions which can potentially capitalize on the advantages of the combined techniques. There has been a significant amount of research and development of hybrid methods and they are the subject of Chapter 4.

Node Table Organization

TIGER incorporates a node file that maintains the coordinates for each node in a polygon network. For each node, a pointer references the first chain in the chain file which has the node as an endpoint. The records in the chain file point to other chains having the node as an endpoint. The significant feature of the node file is the imposition of a spatial ordering. Peano keys are used in a *node directory* to index the node file records in the same way that the linear quadtree orders the leaf nodes of a quadtree. The location-based operations may be enhanced by using binary search methods to identify relevant nodes. For the point inclusion operation, using the Peano key of a given point allows rapid determination of the network node nearest to that point (i.e., nearest in Peano order). The set of chains emanating from the nearest node are obtained from the threaded chains in the chain file. This set of chains divides the space around the nearest node into neighborhoods, one of which may be the polygon containing the point. If none of the neighborhoods contains the point, then this method has at least identified a reasonable location within the polygon network from which the search for the enclosing polygon may begin.

Similarly, the Peano keys of diagonally opposite corners of a rectangular window (or the rectangular extent of an arbitrary window) define a range of keys and allow the rapid retrieval of all the nodes with keys in that range. These nodes reference a set of candidate chains (hence polygons) some of which intersect the window and some of which do not. Each candidate polygon is intersected with the window and non-zero overlaps are subtracted from the window. This continues until the entire extent of the window is covered. Note that the number of nodes initially retrieved may be zero or that the set of nodes retrieved may not reference all of the polygons that intersect the window. In any case, this approach to windowing localizes the windowing operation to a neighborhood within the polygon network from which search for intersecting polygons may begin. This is superior to the exhaustive testing required by other vector models.

3.6. Thematic Overlay

As discussed in Chapter 2, thematic overlay involves the complete traversal of the structures representing two polygon networks. With a vector representation, the fundamental operation for overlay is that of testing for intersecting polygon edges. Applying Weiler's graph comparison algorithm to two polygon networks is very expensive since the search for intersecting edges requires $O(e^2)$ retrievals, where e is the number of edges in the polygon networks. In general, only the edges in the neighborhood of a particular edge need be tested for intersection. Since the methods that enhance location-based search can also be viewed as defining neighborhoods, they can be used to cut down the intersection search for overlay. This concept is discussed in Section 4.3.3.

Chapter 4

Hybrid Data Structures

4.1. Introduction

The tradeoffs between the vector and tessellation approaches to structuring spatial data have long been the catalyst for research in spatial representation techniques that minimize the tradeoffs and capitalize on the advantages of each model. Current and planned GIS applications involving very large data sets demand effective solutions to the problems of access and manipulation. The notion of combining the vector and tessellation approaches has been recognized by a number of researchers [8, 18, 47, 51, 66, 90].

A major theme of this thesis was cogently asserted by Weber [90] in 1978. He pointed out that the advantages and disadvantages of raster data structures are "nearly perfectly complementary" to those of vector methods. Weber went on to say:

> Therefore it is obvious to make an attempt to combine both concepts in a hybrid data structure -- that means to work in one of the two concepts, as long as it is still economic, and to switch over to the other, as soon as it starts to become economic. The point, at which the switching has to take place, essentially is a function of pixel dimension with gridded representation [raster] -- and of the number of involved features with lineal representation [vector]. ... the transition from gridded to lineal representation is to be considered merely as [the] last step in a process of successive refinement defined by a nesting of squares of different sizes.

The structure that Weber proposed is essentially a quadtree spatial index that references an underlying vector encoded polygon network. Each cell of the index references the spatial objects that are contained in or pass through the extent of the cell. Weber noted that the leaf cells of the quadtree should be much bigger than the pixels of a raster representation of the map, otherwise nothing is gained. Unfortunately, he did not describe how to navigate his *locational* data structure nor did he define the raster and vector representations.

The idea of combining techniques for efficient access to spatial data is not new. Several hybrid approaches to structuring spatial data have been suggested in the literature. To classify, evaluate, and compare these approaches we introduce a hybrid spatial data model as a conceptual tool. We begin by describing the two components that comprise the hybrid model: a tessellation component and a vector component. We then discuss criteria that can be used to guide the decomposition principle of the tessellation component. For each criterion, limiting cases are discussed which yield specific formulations of the model that are discussed in the literature.

Aside from classification purposes, the hybrid model serves also as a practical tool. Independent of the decomposition criterion, we therefore give consideration to the construction, maintenance, and use of the model. To facilitate a discussion of the performance of the spatial operations with the hybrid model, a realization of the model is presented. Then we discuss performance of the spatial operations using the hybrid model and present a cursory cost analysis of an average case. In the last section, we discuss other data structures in the literature that bear some resemblance to our hybrid.

4.2. A Hybrid Spatial Data Model

The tradeoffs between tessellation models and vector models are the impetus for a spatial data model that capitalizes on both storage and processing advantages. A hybrid spatial data model, incorporating aspects of both tessellation models and vector models, is the key to conquering this tradeoff. We define a hybrid model as one in which the limiting cases correspond exactly to either a tessellation model or a vector model and whose intermediate formulations exhibit aspects of both. A divide-and-conquer [2] approach to representing spatial data leads naturally to such a hybrid model.

We begin with a vector encoding of a polygon network that covers an area of interest, which is one limiting case of the hybrid model. A tessellation is introduced to subdivide the area of interest into a set of adjacent tiles. This planar decomposition serves as both a partition of and index to the spatial data. Within each tile, the spatial data is still represented by the vector model. We require that the subdivision of the area can continue to any level of resolution. If the highest resolution is that of the pixel level, then the other limiting case, a raster tessellation model, is achieved. The combination of a vector model and a tessellation model in this manner yields a hybrid spatial data model.

The tessellation component of the hybrid model is the division aspect of the divide-and-conquer approach. The conquer aspect is achieved by storing sufficient information in each tile so that spatial operations can be performed on a *tilewise* (i.e., tile by tile) basis. Each relevant tile or pair of tiles is independently retrieved and processed. The reduction in cost complexity due to tilewise processing is discussed in Section 4.3. In the vector limiting case, there is one tile and the entire polygon network is encoded in one vector data structure. In this case, the spatial operations are performed using vector algorithms. In the tessellation limiting case, each tile reduces to a pixel (or block of pixels) and the polygon network is represented by a raster (or region quadtree). In this case, the spatial operations are performed using tessellation techniques. In an intermediate formulation of the hybrid model, a spatial operation is performed in two steps. First, the relevant tiles are determined using the tessellation component and second, the relevant tiles are processed independently using vector techniques.

From a practical standpoint, gaining anything from divide-and-conquer means that a suitable intermediate formulation of the hybrid model must be determined. A specific intermediate formulation depends on the criterion governing the decomposition of a polygon network into tiles. In this chapter we consider several criteria that may be used to guide the decomposition principle and discuss the type of formulation that arises from applying each one to a vector encoded polygon network. A suitable formulation depends on the nature of the application, especially the type and quantity of data and the types of operations applied to the data. We define suitable to mean a formulation in which the management overhead of the tessellation component does not overwhelm the reduced complexity of tilewise vector processing.

4.2.1. Tessellation Component

The decomposition principle of the region quadtree is the basis of the tessellation component of the hybrid model. The quadtree subdivision scheme is attractive for two reasons:

- It is a regular tessellation in which all cells have the same shape and orientation.
- The tessellation is *unlimited* [77] allowing any resolution to be achieved.

The tessellation component is thus a quadtree partitioning of the plane into cells which we term *tiles*. The quadtree serves as both a partition and index to the spatial data. The resolution of the decomposition (i.e., the maximum depth of the quadtree) is governed by a *resolution threshold*. For a region quadtree, the resolution threshold requires subdivision until homogenous blocks (possibly pixels) of colors are obtained. For our hybrid model, we consider several resolution thresholds that lead to different intermediate formulations. Discussion of resolution thresholds is deferred to Section 4.2.3, until we describe the organization of the tessellation and vector components.

A common operational context in which the hybrid model may be used is that of representing and manipulating more than one thematic layer for a given study area. An important pragmatic issue then, is how many quadtree indices (or hybrid data structures) should be used. There are several alternatives: one quadtree encompassing all themes, one quadtree per theme, or some mixture of these.

The advantage of using only one quadtree for all themes is that only one index needs to be stored and maintained and only one index needs to be searched to perform the spatial operations. Within each tile, the themes may be independently represented (an example of this approach is mentioned in [36]) or they may be merged and represented collectively (two examples of this approach are found in [18, 43]). The disadvantage of the second approach is that retrieval of specific thematic data for a study area implies retrieval of irrelevant data since themes are merged. This increases the retrieval cost of the attribute-based operations. Merging themes produces simple and complex polygons with each polygon described by several attributes. This approach necessitates two types of overhead for any query involving a subset of the themes: search within the underlying data structure to locate regions of the relevant color(s) and merging of these regions (by deletion of boundaries) to construct the relevant polygons. This is expensive for all of the attribute-based operations since they typically involve a small subset of the attributes of a theme. This approach also increases the cost of the location-based operations since merged themes must be processed rather than individual themes.

Using one quadtree per theme requires less storage space per tile and allows retrieval of specific thematic data without merging overhead. With this approach, each theme may be individually accessed and only relevant thematic data pertaining to a specific study area need be retrieved. The practical implications of this approach are that different themes must be represented by the same type of data structure and that all of the data structures should have a common origin. Conversion processing is required if either of these requirements is not met. The approach of one quadtree per theme also has the benefit of ease of maintenance which is important in dynamic applications. Updates to a theme require the least amount of retrieval and processing costs compared to the previous approach. Another motivation for this approach, which is a common organizational consideration, is that different themes are owned, stored, and maintained separately. Thus, we choose this approach for our discussion. The foregoing discussion illustrates an example of an issue at a level of organization that is usually higher than that of data structure development. As mentioned in the concluding section of Chapter 1, this type of consideration is not within the scope of this thesis.

We now describe the organization of the quadtree component of the hybrid model. We assume that the polygon network for a theme has been repeatedly subdivided into a set of tiles that meet some resolution threshold. Due to the benefits mentioned in Section 2.3, we choose the linear quadtree to encode the organization of the set of tiles and thematic content of each tile. We note that other quadtree formulations can be used, although they may be less effective. To review, each node in the linear quadtree corresponds to a tile in the tessellation and is represented by a 3-tuple of <key,level,color>. Each tile is identified by a locational code (i.e., key) derived from the coordinates of the lower left corner of the tile and each tile has an associated depth (i.e., level) within the quadtree.

For the purposes of the hybrid data structure, we extend the linear quadtree definition in two ways. Since a tile may correspond to an area containing more than one polygon, a single color designation is not sufficient to encode the thematic content of the tile. Therefore, the color in the 3-tuple is replaced by a list of colors contained within a tile. The second extension is that a pointer is used to reference the vector data describing the interior of the tile. This extended definition yields nodes of the quadtree that are described by a 4-tuple of <key,level,color,data>. The elements in the 4-tuple are:

- key: the locational code of lower left corner of the tile,
- level: the depth of the tile in the quadtree,

- color: a list of or a pointer to the set of colors of polygons within the tile,
- data: a pointer to the vector data structure for the tile.

There are several alternatives regarding the choice of a structure used to realize a linear quadtree (see Sections 2.3 and 2.4). Since the choice does not affect the subsequent discussion, we defer making the choice to Section 4.3, where a discussion of costs depends upon a specific implementation.

4.2.2. Vector Component

The vector component of the hybrid model is the organization of the spatial data within the tiles described by the tessellation component. The choice of model used to represent tile interiors most often depends on the application and the source of the data. The choice between a vector or tessellation method can also be made depending on the properties of the data (e.g., for dense data use a raster, otherwise use a vector method). More research is required in the area of spatial data complexity (e.g., [19]) before the latter criteria can be practically used to guide the choice of tile encoding. The primary motivation for using a vector format within tiles is to define a hybrid model mainly for the purpose of classification (this being the main theme of the thesis) of those data structures which are hybrid in nature, that is, possessing properties of both tessellation and vector models. It is therefore essential that a vector format be used within tiles. The secondary motivation arises from two observations. First, there exist many thematic applications incorporating vector data. Second, for thematic data, a vector representation has two main advantages over a raster representation: it is typically much more compact and the attribute-based operations can be performed more effectively.

Regardless of the vector method used, in order to achieve the conquer aspect of the hybrid model, there must be sufficient information encoded within each tile to enable geographic data to be processed locally within a tile. The vector method used must effectively support the representation of simple, complex, and compound polygons. The representation of the latter type is especially important since the partition imposed by the tessellation on the polygon network introduces compound polygons. We choose the vector representation due to Weiler [91] for two reasons (see Section 3.4). Its hierarchical representation of polygon organization mirrors that of thematic polygon networks and the graph comparison algorithm developed by Weiler has general applicability with respect to the operations we are interested in. Again we note that other vector representations can be used but they may be less effective.

Each tile in the hybrid model represents a subset of a polygon network. Each subset consists of a set of regions corresponding to portions of the polygons that overlap the tile. Polygons which are contained within the tile are made up of *closed* regions, while polygons which are partitioned by the tile boundary are made up of *open* regions. This distinction is important because the spatial operations we are considering can be applied within one tile or across several tiles. Tile boundaries form pseudo-edges (or chains) on the boundary of open regions. Pseudo-edges must be ignored or deleted when considering operations across tiles (e.g., windowing). Weiler's vector representation can easily be modified to allow designation of pseudo-edges. Pseudo-edges can be used or ignored as required, but they must be identified. The identification of edges as pseudo-edges is the only modification we make to Weiler's graph data structure (see Section 3.4).

4.2.3. Resolution Thresholds

A very important characteristic of the hybrid model is the resolution threshold that governs the quadtree decomposition. The type of threshold and its particular value determine several parameters describing the resulting data structure. These parameters are the size and depth of the tessellation component as well as the *tile capacity* of the vector component (i.e., the amount of data in each tile). These parameters directly affect the efficiency of the hybrid model for performing location-based and attribute-based operations. In this section, we consider several resolution thresholds as applied to the decomposition principle of the region quadtree.

We begin by making a general comment that applies to all of the resolution thresholds. By selecting a suitably large value, any of the thresholds yields a single node in the tessellation component. This represents a single tile that covers the entire area of interest and the hybrid structure is basically a vector representation. In this case, the tessellation component may be eliminated.

At the other extreme, the quadtree decomposition can continue indefinitely. In reality, there are practical considerations that constrain quadtree decomposition to finite levels. Physical constraints of data capture limit the amount of decomposition for any tessellation structure, beyond which further decomposition would not yield additional useful information. We refer to this maximum resolution of a tessellation as the *pixel level* since the smallest allowable cells are termed pixels. Note, on the other hand, that the finite machine representation of numbers effectively introduces another grid which constrains the vector component since coordinates of vertices are defined using this finite number system. We must assume that the machine resolution is at least as fine as the resolution of the pixel level, otherwise all pixels could not be represented. Conceptually, this means that at the pixel level, a polygon network does not necessarily consist of uniform pixels. For example, a pixel may contain a polygon network node at which several edges are incident. Such a pixel does not have one color as an attribute since it is shared by more than one polygon. In practice, the implementation of a tessellation data structure representing such a situation usually requires a decision regarding the color assigned to the pixel. The decisions made at the pixel level for each resolution threshold discussed below, yield limiting cases of the hybrid model that correspond to specific data structures described in the literature.

In general, a tile corresponds to an area covered by a subset of a polygon network. At the pixel level, a tile corresponds to a pixel representing one of three situations in a polygon network. The pixel may be contained entirely within one polygon. The second case is that of a pixel in which one or more edges pass through it. The third case involves a pixel in which one or more vertices fall within its extent. It is the latter two cases, in which a pixel is shared by more than one polygon, that data within the pixel may need to be adjusted (approximated) in order that the governing resolution threshold be satisfied.

For any of the resolution thresholds we consider, we require that all tiles above the pixel level (i.e., tiles that are not of pixel size) satisfy the threshold value. At the pixel level, a threshold such as a maximum number of colors per tile, may not be achieved. We allow tiles at the pixel level to violate the resolution threshold guiding the decomposition. We will, however, describe methods of adjusting the vector component of pixel size tiles to ensure that the governing resolution threshold is satisfied here as well. These modifications will be shown to yield specific spatial data structures described in the literature, thus demonstrating the general nature of the hybrid model.

Before proceeding with the discussion of resolution thresholds, we should comment on the practical considerations that can influence the representation of thematic data with the hybrid approach. Application dependent criteria often determine the tile

THE

ANAD

SES

IENNES

decomposition rather than one of the resolution thresholds discussed below. A common consideration of agencies that deal with thematic data is the existence of one or two standard map sizes. Typically, the vast majority of spatial operations can be resolved by considering one or a few maps. In such cases, it is reasonable to fix the tile size to that of the most utilized map series [4]. If most spatial operations involve more than one map then it may be desirable to select a larger tile size which aggregates a standard number of maps. A tile decomposition based on heavily used map series is both intuitive and cost effective since the least number of tiles are retrieved for the majority of operations. There are other application dependent criteria and properties of the data that can affect the choice of tile resolution to best satisfy application requirements. Two common examples are any known trends in data density and the size and location of the most requested windowing areas. It is desirable to have a tile size in which the densest tile can be retrieved with an acceptable number of disk accesses. As we shall see, this allows the most efficient performance of the spatial operations.

Tile Area

A simple criterion controlling resolution is that of tile area. Decomposition continues until each tile has an area less than or equal to an area threshold a. This criterion gives a tessellation consisting of tiles of uniform area and uniform depth. This criterion can always be satisfied by the decomposition principle of the region quadtree. If the area threshold is that of a pixel, then the tile decomposition is that of a raster grid. If we require that each tile have at most one color and if we eliminate the vector component for each tile, then the resulting structure is in fact a raster representation. In particular, the raster is represented by a list of pixels ordered (or indexed) by locational code.

For area thresholds larger than pixel size, the tessellation component always consists of a list of nodes. Since updates to the thematic data affect only the vector component, the tessellation component is static. This allows retrieval of nodes in the tessellation component through the use of direct access techniques. Tiles of uniform area means that the tile capacity is variable. An effective way to realize this approach is to use a file (expandable bucket) to store the vector component of each tile. The pointer element of the 4-tuple representing a node in the tessellation component can then consist of a file pointer. This method allows convenient use of the file handling mechanisms found in operating systems for accessing the vector component.

Number of Colors / Number of Polygons

This threshold is essentially that which governs the region quadtree. Tiles are subdivided until each tile contains c or fewer colors, where c is the color threshold. Similarly, tiles may be subdivided until each tile contains p or fewer polygons, where p is the polygon threshold. Since several polygons within a tile may have the same color, this second threshold produces similar, but not identical, tessellations. If p denotes compound polygons, then both thresholds yield equivalent tessellations. If p denotes simple polygons, then for equal values of p and c the polygon threshold yields a finer decomposition than the color threshold.

In the limiting case, we consider a threshold value of $c=1$ (or equivalently $p=1$) for the remainder of this section. Neither of these threshold values may be satisfied at the pixel level. Depending on the decision made at the pixel level, to ensure satisfaction of either threshold, one of three known data structures arises.

The simplest decision is not to modify the tessellation component thus allowing multi-colored pixels. If the vector component for each tile is eliminated, then the resulting structure is a generalization of a quadtree formulation proposed by Hunter and Steiglitz [37]. They address the problem of representing simple polygons by using a three-color variant of the region quadtree. Their formulation, termed an *MX quadtree* by Samet [77], considers the boundary of a polygon as separate from both the interior and exterior of the polygon. An MX quadtree contains nodes of three types - interior, exterior, and boundary. A boundary node is a node which is intersected by a polygon edge. An image containing a polygon is repeatedly subdivided until all boundary nodes are pixel sized (see Figure 4.1). Interior and exterior nodes, corresponding to areas within and outside a polygon, are aggregated as long as they are homogenous (as with the region quadtree).

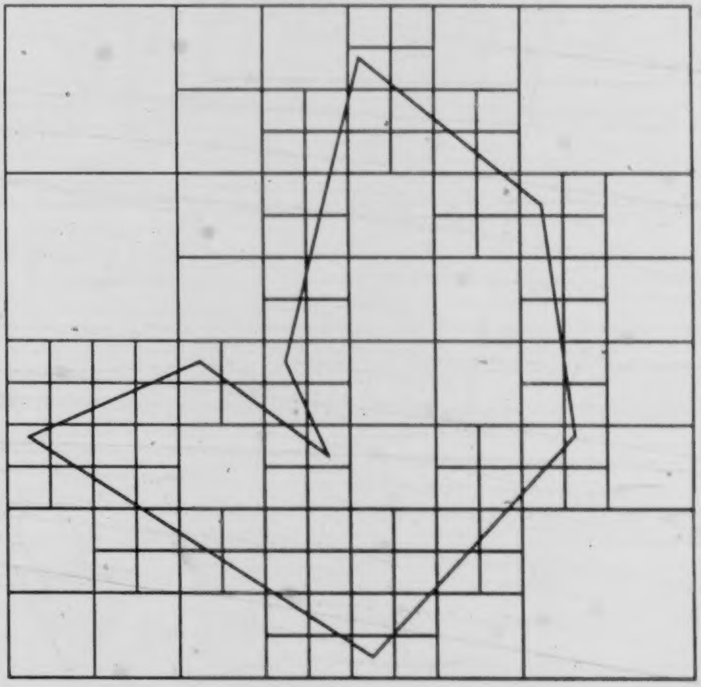


Figure 4.1 MX Quadtree Representation of a Polygon.[4]

---

[4] Modified from Samet [77].

The current formulation of the hybrid model yields a multi-color generalization of the MX quadtree. Nodes of a single color represent interior and exterior nodes while multi-color nodes represent boundaries and vertices. Boundaries of a polygon network are approximated at the pixel level, by assigning a non-zero width to polygon edges and by assigning single pixels to vertices. In comparison to vector models, the MX quadtree may require substantially more storage since boundaries are approximated at the pixel level.

The color threshold is satisfied for tiles larger than pixels, otherwise the tiles would have been decomposed further. The second data structure arising from a color threshold of $c=1$ is obtained by approximating the data in order to ensure that the threshold is also satisfied at the pixel level. For each multi-color pixel, the approximation consists of selecting one color from the list of colors in the node. If we aggregate sets of like-colored pixels (or blocks) that result from this color selection and if we eliminate the vector component for each tile, then we obtain the region quadtree formulation.

The third data structure is obtained exactly as before (set $c=1$ and select a single color for each multi-color pixel), but now, in addition, the vector component is retained for each tile, with an appropriate modification. Enforcing the color threshold for a pixel size tile requires adjustment of the vector component of the tile to reflect the boundaries pertaining to the chosen color. This means that the four orthogonally adjacent tiles must be examined and possibly adjusted to reflect the chosen color. The resulting data structure is a region quadtree which incorporates boundary information in its leaf nodes. This is almost exactly the *line quadtree* formulation proposed by Samet and Webber [76].

The line quadtree is a hierarchical approach for representing both the areas and boundaries of individual regions comprising polygonal maps. The line quadtree attempts to alleviate a deficiency of the region quadtree, namely, the lack of border information.

This is achieved by encoding boundary information within the nodes of a region quadtree. A regular decomposition is used until blocks (possibly pixels) are obtained that have no line segments passing through their interior. This is equivalent to the region quadtree since lines separate regions of different colors. Each leaf node of a line quadtree stores border information by encoding which of the four sides of the leaf correspond to a complete edge of any region. Border information is hierarchically maintained in that non-leaf nodes also store border information. Each non-leaf node indicates the presence of an edge corresponding to one of its sides as long as the edge is uninterrupted by T-junctions with any edges encoded within its descendents. Figure 4.2 shows the line quadtree representing the example polygon network of Figure 1.1. Solid lines indicate the presence of boundary information while dashed lines indicate the absence of boundary information. The difference between Samet and Webber's line quadtree and our hybrid formulation is that the former stores boundary information in both internal and leaf nodes while the latter, being a linear quadtree, stores boundary information in only leaf nodes. Note that, had we chosen a pointer-based realization for the tessellation component of our hybrid model (i.e., with internal nodes), the line quadtree formulation could be achieved exactly.

The line quadtree requires that all regions be rectilinear so that edges eventually overlap only sides of nodes, not node interiors. Polygon networks describing thematic data are typically composed of irregular regions. Using the line quadtree to represent such polygon networks is undesirable for two reasons. Since regions are irregular, the decomposition typically must continue to the pixel level to yield leaf nodes whose sides correspond to edges. Furthermore, all of the non-leaf nodes do not contain edge information since the sides of large quadtree blocks do not correspond to edges. The hierarchical storage of border information in the line quadtree enables efficient techniques for boun-

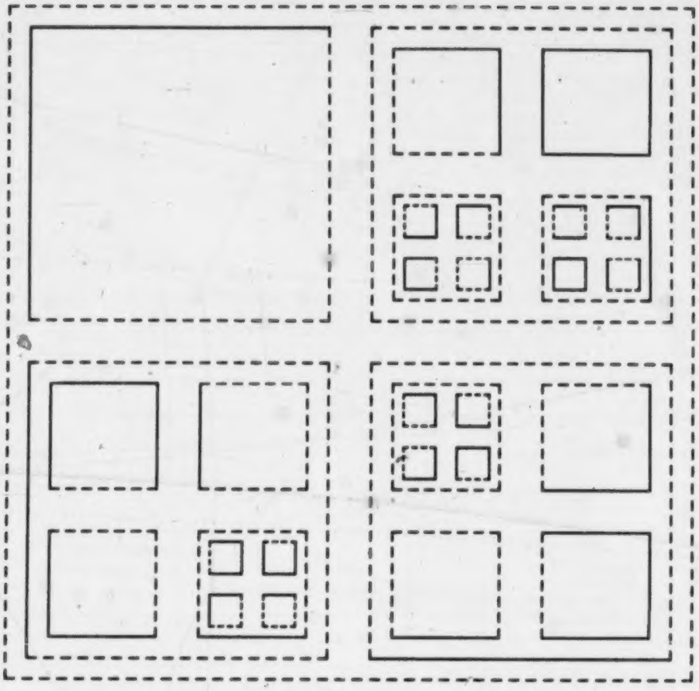


Figure 4.2 Line Quadtree for the Polygon Network of Figure 1.1.

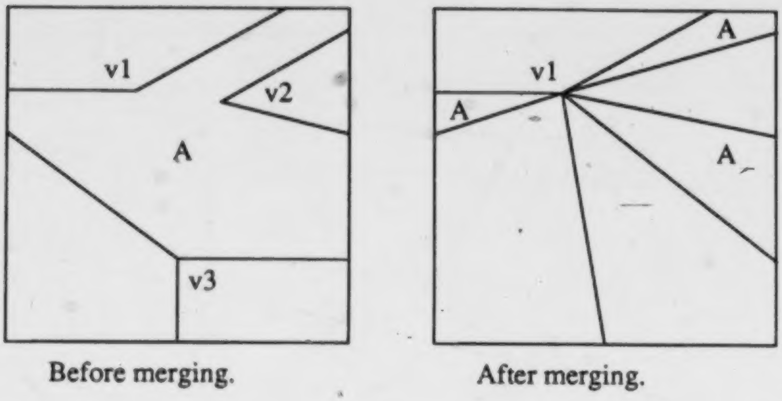
dary following and thematic overlay [77]. While this may be true for rectilinear maps, it appears to be less successful for thematic applications.

The line quadtree and the MX quadtree are methods that may be used for the representation of polygon networks. The decomposition induced by the line quadtree for irregular thematic data is similar to that induced by the MX quadtree since boundaries are usually represented at the pixel level. However, the line quadtree does not associate a width with boundaries as does the MX quadtree. In general, both structures correspond to approximations of a map [77]. The MX quadtree represents edges by BLACK pixels while the line quadtree requires that all regions comprising a map be rectilinear which means, for example, that the situation of five regions sharing a vertex must be approximated at the pixel level. The serious drawback of both variations is that, for thematic data, they yield very deep quadtrees since edges are usually approximated at the pixel level [77].

Number of Vertices

This threshold involves the vertices of a polygon network. In this approach, the study area is repeatedly subdivided into quadrants until the number of vertices in each tile is less than or equal to a vertex threshold v. There are two ways of specifying the vertex threshold. The threshold value may refer to the number of vertices originally present in the network. Alternatively, the value may refer to vertices originally present in the network plus the vertices on the tile borders introduced by the partition. Both methods lead to tiles containing a variable amount of vector data. The first method does this since the number of vertices introduced by the partition is variable. The second method yields tiles with a fixed number of vertices but still variable amount of vector data since the degree of each vertex is variable. As with the previous threshold, both specifications of the vertex threshold yield tiles of varying extent. We choose the number of original vertices threshold for the remainder of this section since the other specification has no advantages.

For a tile at the pixel level that violates the original vertex threshold, the threshold may be enforced by an operation local to the tile. If $v=1$, then one vertex is selected arbitrarily to represent the tile. The remaining vertices are merged with the selected vertex by adjusting their incident edges to terminate at the selected vertex. By not moving the tile boundary intercepts of any edges, this operation is confined to the current tile. Figure 4.3 illustrates the vertex merging operation in which vertices $v2$ and $v3$ are merged with vertex $v1$.



Before merging. After merging.

Figure 4.3 Example of Vertex Merging.

This operation effectively approximates the vector component of a tile but only at the pixel level. Any threshold $v = k$ can be satisfied by this operation by repeatedly merging vertices until k remain.

If we choose the threshold value $v = 1$, then this approach leads to a formulation developed and analyzed by Samet and Webber [78]. They have developed a quadtree for storing polygonal maps that is based on an alternative of the region quadtree that associates point data with quadrants. This alternative, termed a *PM quadtree* (polygonal map), is based on an earlier formulation known as a *PR quadtree* (point region) [77] that was developed for the representation of point data. In the PR quadtree, a regular subdivision is applied until leaf nodes are either empty or contain a single point and its coordinates. The PM quadtree is the result of further adapting the PR quadtree to store polygon boundary information. The leaf nodes of a PM quadtree partition the edges of a polygonal map into partial edges that either span an entire block or enter and terminate at a vertex within a block. These partial edges are termed *q-edges* and for every leaf node they are divided into seven classes. One class corresponds to the q-edges that meet at a vertex within a block while the other six classes denote q-edges that enter a block at one side and exit at another. A tree structure is used to order and store the q-edges within a leaf node by class. Associated with each q-edge is a pair of names indicating the polygons on either side. Figure 4.4 shows an example polygon network and its PM quadtree.

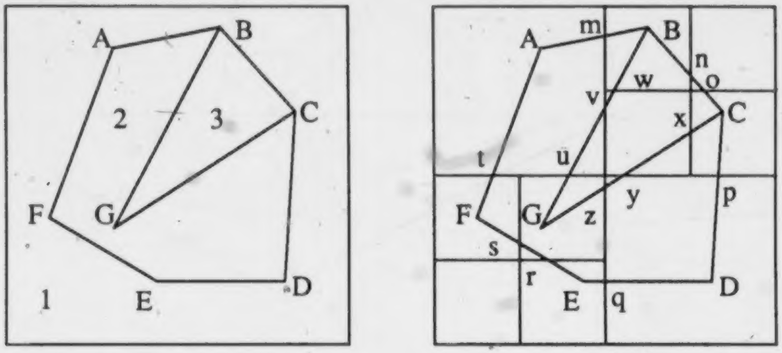


Figure 4.4 A Polygon Network and its PM Quadtree.[5]

The q-edges of the PM quadtree are analogous to the vector component of our hybrid formulation. Examples of q-edges in Figure 4.4 are segments xy and qD.

The PM quadtree is an attempt to resolve some of the deficiencies of the MX quadtree and the line quadtree. These deficiencies are the pixel level approximations of polygon boundaries and vertices by the MX quadtree, and the requirement of rectilinear regions by the line quadtree. The PM quadtree formulation does not have these deficiencies and is a convenient, reasonably efficient data structure for performing the operations we are interested in [77, 78], for polygonal maps of *reasonable* size. The main drawback of the PM quadtree (and the recent variation of it called a *PMR quadtree* [79]) is the use of a vertex threshold value of one. For very large spatially referenced data sets, partitioning to one vertex per tile is too extreme and results in a very large quadtree. For irregular thematic data, where there are a large number of vertices, the PM quadtree effectively converts vector data to a tessellation format.

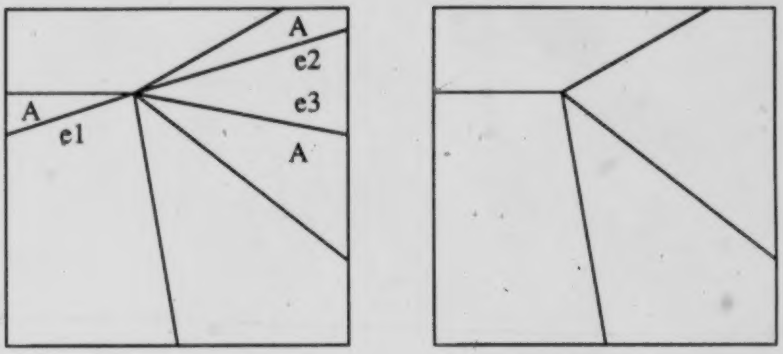
---

[5] Modified from Samet [77].

Number of Edges

With this threshold, subdivision continues until the number of partial edges in each tile is less than or equal to an edge threshold e. In general, a specific value for this threshold cannot always be satisfied when a polygon network contains vertices of degree n and $n > e$. As with the vertex threshold, there are two ways to specify the edge threshold. The threshold value may refer to the number of partial edges originally present in the polygon network or the value may refer to the partial edges originally present in the network plus the pseudo-edges introduced by the partition. Both methods of specifying the edge threshold lead to tiles of variable capacity. This is because the number of pseudo-edges introduced by the partition is variable and the number of polygons defined by the set of all edges in a tile is also variable. Both specifications yield tiles of varying areas. In the following discussion, e is the threshold for partial edges originally present in the network.

Enforcement of an edge threshold at the pixel level may be accomplished by an operation that begins with vertex merging as described earlier. Vertex merging is used to reduce the number of vertices to one. Then partial edges are repeatedly merged until the threshold is satisfied. Edge merging may be viewed as coalescing two neighboring edges into one by merging their endpoints on the tile boundary. This operation reduces the number of edges and number of regions in the tile by one. Figure 4.5 illustrates the edge merging operation for the tile shown in Figure 4.3. In Figure 4.5, edges $e1, e2, e3$ are merged with adjacent edges so as to satisfy an original edge threshold of $e=4$. Since this operation effectively removes polygons within the tile (e.g., the regions labelled A in Figure 4.3), the polygon networks of one or more of the eight adjacent tiles may be affected.



Before merging. After merging.

Figure 4.5 Example of Edge Merging.

The edge merging operation, while not localized like the vertex merging operation, affects only adjacent tiles since the vector data is adjusted only on the boundary of the current tile. Since edge merging reduces the number of edges in a tile, as well as adjacent tiles, a single pass of all pixel level tiles violating an edge threshold is sufficient to satisfy the threshold. A minor effect of the edge merging operation is that remaining edges may end up separating regions of the same color, in which case they may be deleted.

While edge merging can be used to satisfy any edge threshold, we note that low threshold values may introduce unacceptable distortions in a polygon network. For example, an edge threshold of $e = 1$ does not permit the representation of three common occurrences: more than one partial edge in a tile, more than one vertex in a tile, or a combination of the previous two cases. This threshold does not permit the proper representation of a node of degree greater than two. Note that in the vicinity of a node of high degree there exist pixel size regions which contain two or more partial edges. Satisfaction of $e = 1$ requires approximating vertices and edges with pixels. With this deficiency in mind, we now discuss two quadtree formulations that arise from using an edge threshold.

The *edge quadtree* of Shneier [80] repeatedly subdivides a region containing linear data into quadrants until squares are obtained that contain a single curve that can be approximated by a single straight line. This formulation effectively uses an edge threshold of $e=1$. The leaf nodes store information about edges including direction and intercept, and in the case where an edge terminates within a node, the coordinates of the endpoint. The edge quadtree requires fewer nodes than the MX quadtree or the line quadtree to represent the same polygon network. Long edges may be represented by large leaves or sequences of large leaves, in sharp contrast to the MX and line quadtrees. Small leaves are required in the vicinity of areas of high curvature (e.g., corners) and network nodes since these areas contain two or more partial edges. Figure 4.6 shows the edge quadtree for the same polygon used to illustrate the MX quadtree in Figure 4.1. Samet [77, 78] has noted the serious problem of the edge quadtree representation in that it cannot represent two or more edges emanating from a vertex except as a pixel corresponding to an edge of minimum length. Two drawbacks stemming from this inadequacy are that all vertices of a polygon network are stored at the pixel level resulting in deep quadtrees and that boundary following cannot be properly handled.

Closely related to the edge quadtree is the *least squares quadtree* of Martin [55]. The leaf nodes of a least squares quadtree contain curves that can be approximated by k straight line segments within a least squares tolerance (note that this differs from the PM quadtree since the latter formulation represents edges exactly). This formulation effectively uses an edge threshold of $e=k$. Since k can be selected based on the complexity of a polygon network, this approach is superior to the edge quadtree. The least squares quadtree requires less nodes than the edge quadtree for a given polygon network.

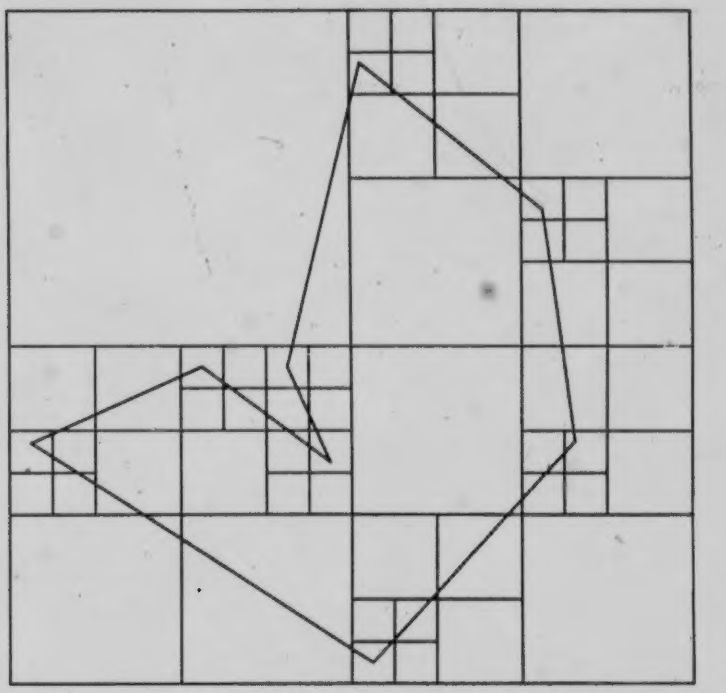


Figure 4.6 Edge Quadtree for the Polygon of Figure 4.1.[6]

Note that the least squares quadtree can provide a better approximation than the edge quadtree for tiles containing nodes of degree greater than *k*.

Tile Capacity

The previous three thresholds (colors, vertices, and edges) generally yield tiles of varying extent which is more storage efficient than fixed area tiles when the density of the data is not uniform. However, these thresholds yield tiles of variable capacity since, respectively, the number of edges comprising a polygon is variable, the degree of a vertex is not fixed, and a fixed set of edges can define a variable number of polygons. Tiles of variable capacity implies an increased cost of tile retrieval compared to fixed capacity tiles. Two observations allow us to claim that this deficiency is not debilitating. First, we observe that a specific edge threshold implies a bound on the amount of vertex data in a tile. The second observation stems from the well-behaved nature of thematic data found in practical applications. It is possible to estimate reasonably the number of polygons defined by the set of edges within a tile. Other researchers have also made this observation for practical settings [24, 79].

---

[6] Modified from Samet [77].

Consequently, it is reasonable to assume tiles of bounded capacity arising from these tile thresholds. We can ensure fixed capacity tiles by using as a threshold, the total storage space required by the vector component for each tile. In essence, this threshold is a combination of the color, vertex, and edge thresholds. This capacity threshold requires repeated decomposition until the space required by the vector component of each tile is less than b bytes, for example.

A fixed-size bucket method may be used for the vector component of the hybrid structure when fixed capacity tiles are achieved. The data element of each tessellation component node references the location of a bucket. In the case of pixel level tiles violating the capacity threshold, chaining of overflow buckets may be used, or the data can be reduced to satisfy the threshold. Again, we note that for well-behaved thematic data sets, violation of the capacity threshold is expected to occur very infrequently. Examples of data structures based on a tile capacity threshold may be found in [18, 63].

4.2.4. Updating The Hybrid Data Structure

A dynamic environment implies that the hybrid structure must be updated to reflect additions, deletions, and modifications to a polygon network. Within a tile, the vector methods discussed in Section 3.4 can be used to update the tile interior. These updating operations however, can affect the amount of data represented in the tiles. Since most of the thresholds discussed in this chapter relate to the amount of data within tiles, splitting and merging operations must be available to maintain the hybrid structure at a specified threshold value (a range of values may be more effective for dynamic applications).

A tile is split into four sub-tiles whenever a resolution threshold is exceeded (e.g., number of vertices). This operation requires splitting the polygon network within the tile into four polygon networks, one for each sub-tile. This can be accomplished by extracting four windows consisting of the four quadrants in the tile. Weiler's graph comparison algorithm can be used to effect windowing by defining the windows as polygons in another theme and intersecting each window with the original polygon network. The result of the intersection with each window produces a polygon network in the same vector format as that of the vector component. Of concern in tile splitting is the creation of open regions from previously closed polygons. Some polygons are unaffected by a split and they are simply moved to one of the sub-tiles. Polygons split by the quadrant lines create open regions in the relevant sub-tiles. The issue here is to ensure that resulting open regions are properly identified with pseudo-edges in the vector component of the sub-tiles.

The split operation affects the tessellation component in that the node corresponding to the split tile is replaced by four child nodes. For each child node, the elements of the 4-tuple describing the node are assigned as follows. The key element is set to the locational code of the southwest corner of the sub-tile. The level element is one greater than the level of the parent node. The color element is adjusted to include only those colors appearing in the sub-tile. The data element is set to refer to the location of the polygon network for the sub-tile. The split operation adds tiles to the vector component which implies that new buckets or files may be required to store them.

The converse of splitting is merging and it is used to aggregate four tiles when a lower limit of a resolution threshold is passed. Merging is accomplished by linking the binary trees of the sub-tiles to construct the binary tree of the parent tile. For those polygons that cross the sub-tile boundaries, the graph data structures of the constituent regions are adjusted to connect the regions. This is done by linking the edge-side loop of a region with the edge-side loop of the corresponding region in the adjacent sub-tile. The pseudo-edges within the edge-side loops are deleted and the resulting gaps are closed by connecting edges with matching coordinates (on the sub-tile boundaries). Weiler's graph comparison algorithm may be used to effect merging by using it with four sub-tiles and performing a union operation.

The merging operation affects the tessellation component in that the four nodes corresponding to the merged sub-tiles are replaced by one node. The elements of the 4-tuple describing the new node are assigned as follows. The key element is the locational code of the southwest corner of the southwest sub-tile. The level element is one less than the level of the child nodes. The color element is the union of the color lists of the four child nodes. The data element is set to refer to the location of the merged polygon network. The merging operation implies that some tiles are deleted which means that the buckets or files storing those tiles may also be deleted.

Merging of tiles is not necessarily required since it is irrelevant whether a tile consists of one polygon network or of four sub-tiles. Merging is not required since the spatial operations can be performed without change. Pseudo-edges are used, ignored, or deleted as required if the operation spans more than one tile. The independent processing of relevant tiles allows merging to be an optional function. A consequence of this is that the tile hierarchies of the input hybrid data structures are reflected in the tile hierarchy of the output hybrid data structure. For example, a window extracted from *n* tiles will consist of *n* tiles.

The splitting and merging operations may also be required after performing some of the spatial operations. In particular, the set operations and the thematic overlay operation can yield polygon networks that are too dense or too sparse for the resolution threshold value in use. Either the threshold value for the resulting hybrid structure is ignored or adjusted to suit the result, or the result is split/merge processed to meet the current threshold.

4.2.5. Constructing The Hybrid Data Structure

Construction of the hybrid data structure must be possible if the structure is to be useful. The source data for GIS applications can exist in either a tessellation format or a vector format or it may reside on conventional maps. The important issue for construction of the hybrid is to have the data in a vector format since that is the basis of the hybrid. There are two methods that may be used to build the hybrid structure. The appropriate method depends largely on the current organization of the data. A top-down method is used if the data is organized globally (i.e., it is completely aggregated). A bottom-up approach can be used if the data is originally partitioned in some manner (e.g., a digital map series) or if the data is to be digitized from maps.

The top-down method begins by considering the entire data set as one large tile and then consists of repeated application of the tile splitting operation. As sub-tiles are created, the nodes of the tessellation component are created and inserted into a linear quadtree structure. Tile splitting occurs until a resolution threshold is satisfied.

The bottom-up method consists of treating the individual maps as tiles in the final hybrid structure. The vector component for each tile is searched to discover its size, location and color content to create a node for the tessellation component. As each node is created, it is inserted into a linear quadtree structure. This method requires maps that are uniform in size or that the organization of the maps is congruent to a quadtree decomposition (note that maps can often be split or merged to achieve this congruency). If the maps exist only as conventional physical products, then this is the preferred method for constructing the hybrid structure in conjunction with digitizing of the maps. The merging operation may be applied to groups of tiles that are too sparse if a resolution threshold is desired. Similarly, tiles may be split if they are too dense.

4.3. Spatial Operations

In this section we consider the performance and the cost of the hybrid model for performing the spatial operations introduced in Chapter 1. To do this, we make a number of assumptions about and discuss the issues related to the implementation of the hybrid data structure. Then we describe the location-based and attribute-based operations and comment on their cost complexities. Finally, the thematic overlay operation is described and a cursory cost analysis is presented to demonstrate the divide-and-conquer advantages that may be achieved by using the hybrid method.

The major assumption of this thesis is that we are dealing with very large thematic data sets. As a consequence, it is assumed that both the tessellation and vector components of the hybrid model reside in secondary storage. Since specific statements are to be made about retrieval costs using the model, further requirements (in addition to those already imposed in Section 4.2) concerning its actual realization need to be specified.

The tessellation component is assumed in Section 4.2 to be represented by a linear quadtree. However, as noted in Sections 2.3 and 2.4, there are still numerous ways by which a linear quadtree may be stored and accessed in secondary storage. Since the tessellation component is fundamentally a location-based structure, the goal is to select a realization that offers the best location-based access.

The cell methods discussed in Section 2.4 can be used to achieve direct location-based access to the nodes of the tessellation component. In particular, the EXCELL method guarantees retrieval of a node with two disk accesses at the expense of a potentially large directory. However, for reasons given in the next paragraph, a large directory may not be a problem. If it is, then the cell method due to Davis and Hwang [22] may be employed to keep the directory to a manageable size and offer expected, but not guaranteed, $O(1)$ location-based access. For the purpose of this section, we choose the EXCELL method to realize the tessellation component. The main reasons are the simplicity of the method and guaranteed direct access based on location. Using EXCELL to realize the tessellation component means that each bucket stores more than one quadtree node. While EXCELL is typically used for storing fixed-length nodes, it is practical also for storing the variable-length nodes (since there are multiple colors per node) of the tessellation component using a reasonable bucket size. Note that each node in the tessellation component references one tile in the vector component.

An important point about the relative size of the tessellation component should be noted. In general, we are dealing with data sets that are massive if represented entirely by a tessellation method. However, the depth of the quadtree in the hybrid model typically is not at the pixel level. The quadtree is "cut-off" at a level which permits a reasonable amount of vector data to be retrieved with $O(1)$ accesses. This makes the tessellation component quadtree much smaller than the corresponding region quadtree for the entire data set. The directory used by the EXCELL method contains pointers that reference a level above the nodes of the tessellation component quadtree. Thus, the EXCELL directory has significantly fewer elements than the number of nodes in the tessellation component quadtree. The tessellation component typically introduces a small amount of storage in addition to the storage required for the vector component. For a quantitative example, see Appendix A1.

As discussed in Section 4.2, the vector component of the hybrid model is realized by the polygon-driven structure due to Weiler [91]. Each tile consists of a Weiler data structure which requires $O(p+e)$ space, where p is the number of polygons and e is the number of edges in the tile. It is desirable that a tile be retrieved with $O(1)$ disk accesses. To facilitate this, the resolution threshold governing the tile decomposition should yield fixed capacity tiles. Then, a bucket scheme can be used to effect retrieval of a specific tile with one disk access. With variable capacity tiles, the situation is more complicated. If a bucket size is selected which accommodates the tile of highest capacity, then each tile resides in at most one bucket. This bucket size allows for the storage of variable capacity tiles and for the possibility of storing multiple small tiles per bucket. Provided that buckets of reasonable size can be used for the storage of the tiles, we expect that, having obtained a node in the tessellation component, we can retrieve the corresponding tile with one disk access. Thus, the total retrieval cost for retrieving a specific tile is assumed to be $O(1)$ disk accesses.

The general method for performing all of the spatial operations under consideration consists of two phases. The first phase involves only the tessellation component and consists of determining the set of tiles relevant to the operation. For the location-based operations, where one hybrid structure is involved, this phase determines one set of individual relevant tiles. For the attribute-based operations and thematic overlay, where two hybrid structures are involved, pairs of relevant tiles are determined. Once the relevant tiles or pairs of tiles are determined, they are independently retrieved. The second phase for each operation consists of applying a vector algorithm to each retrieved tile or pair of tiles. The result of the vector operation is a polygon network represented by Weiler's data structure. As relevant tiles or tile pairs are processed, the nodes for the tessellation component of the resulting hybrid structure are created.

A key aspect of this two-phase method is that there is no recombination cost associated with obtaining the result. For all of the operations, except point inclusion, the resulting hybrid structure possesses a tile decomposition that is related to the input hybrid structure(s). For example, the windowing operation produces a window represented by a hybrid structure with a tile decomposition reflecting that of the input. The overlay operation produces a hybrid structure whose tiles are always the smaller of the two input tiles corresponding to the same portion of the area of interest.

All of the spatial operations are performed on a tilewise basis (i.e., only one tile or one pair of tiles is processed at a time). We assume that the vector components comprising two tiles can be stored in memory simultaneously. The basic task for all of the spatial operations is that of detecting and computing the intersection of two line segments (usually edges). We denote the cost of executing these vector operations as *processing cost*. In summary, the cost complexity of the spatial operations is then dominated by the cost of node and tile retrievals. Typically, the processing cost of the spatial operations is secondary since they occur in memory.

Before proceeding with the discussion of the spatial operations, we introduce some notation and state some assumptions. Unless otherwise indicated, we assume the existence of a theme t that is made up of e edges comprising p polygons. The hybrid structure for t consists of m nodes in the tessellation component and hence m tiles. The region quadtree encoding for t is assumed to have n nodes. For resolution thresholds whereby each tile contains many edges, we assume that $n \gg m$.

4.3.1. Location-Based Operations

A location-based operation with a totally vector representation requires exhaustive traversal of a polygon network compared to the $O(1)$ retrievals required with a tessellation representation. The hybrid model offsets this since the tessellation component allows the vector operations to be localized to relevant tiles.

For the point inclusion operation, the first phase consists of determining which tile contains a given point. This is accomplished by using extendible hashing to determine which bucket contains the relevant linear quadtree node. Two disk accesses are required: one to locate and the other to retrieve the relevant node in the tessellation component. Once the data element of the 4-tuple describing the node is available, the relevant tile is retrieved with one access.

The second phase of the point inclusion operation consists of traversing the retrieved vector component to discover which polygon contains the point. The well known *plumbline* algorithm may be used to test each polygon (and in turn each edge) in the tile. The result of this operation may be the color of the enclosing polygon or a Weiler structure representing the enclosing polygon. Since the vector component for a tile may be stored entirely in memory, the cost of the second phase is $O(e_i)$ comparisons, where e_i is the number of edges in the relevant tile. The overall cost of the point inclusion operation is thus $O(1)$ disk accesses plus $O(e_i)$ comparisons.

In general, the windowing operation consists of extracting a portion of a polygon network. The operation takes as parameters a hybrid structure representing t and a specification of the window (its position and shape). The operation returns a new hybrid structure representing the extracted window. Typically the window is specified by a list of coordinates defining its boundary and we assume that the window consists of a simple polygon that can be stored in memory.

While there are several techniques that may be used to perform windowing, a particularly effective method makes use of the locational code order of the tessellation component nodes. A bounding rectangle around the window can be used to compute the quadtree nodes corresponding to tiles that intersect the rectangle. This technique avoids traversal of the tessellation component of t [38]. This method identifies a set consisting of w relevant tiles. Each relevant tile is retrieved and intersected with the window using Weiler's graph comparison algorithm. The cost of windowing is then $O(w)$ tile retrievals and $O(e_w)$ processing, where w is the number of relevant tiles and e_w is the number of edges in the relevant tiles. The shape and extent of the window over the area of interest determine the value of w, which can equal m in the worst case, but it should be noted that typical windowing operations involve windows that are significantly smaller than the area of interest.

Representing the window with a hybrid structure allows a general window specification in the form of a binary theme. For example, BLACK regions may specify the interior of the window while WHITE regions specify the exterior of the window. The windowing operation then consists of an intersection operation between the BLACK regions of the window hybrid structure and the input hybrid structure t. General windowing specified in this manner is effectively an attribute-based operation.

4.3.2. Attribute-Based Operations

The basic task for attribute-based access is the retrieval of all of the polygons within theme t of a specified color. The two-phase approach for this task proceeds as follows. First, the tessellation component of t must be traversed entirely to discover which nodes reference the relevant color. This traversal yields w relevant tiles and costs $O(m)$ retrievals. The second phase consists of retrieving each relevant tile and extracting the vector data pertaining to the relevant color. The cost of the second phase is $O(w)$ retrievals plus $O(e_w)$ processing cost, where e_w is the total number of edges in the relevant tiles.

For attribute-based access, the tessellation component must be traversed completely so that the location of relevant colors may be determined. However, the cost of traversing the quadtree of the hybrid structure is improved over the cost of traversing a pure tessellation representation since the tessellation component is significantly smaller than a total quadtree representation of the data set. This is because the tessellation component is not subdivided to the pixel level and color information is recorded once per tile. While vector attribute-based algorithms (i.e. set operations) are more complex than their tessellation counterparts, they can be applied to only those polygons with the appropriate colors. In contrast, the tessellation set operations must be applied across entire tessellation structures. The complexity of the vector operations is offset by the reduced number of objects that they are applied to. This is worth noting even though the operations are conducted in memory.

The set operations involve two input hybrid structures t_1 and t_2 representing two themes. Two polygon colors, c_1 and c_2, specify which attributes of each network are relevant. A parallel traversal of the input hybrid structures is used to perform any of the set operations and may be performed as follows.

1. Traverse the tessellation component of both t_1 and t_2 and retrieve the relevant nodes (i.e., nodes containing the colors c_1 and c_2 in their color lists, respectively).
2. Perform the set operation on the sets of relevant nodes retrieved in 1.
3. Retrieve the tiles corresponding to the nodes resulting from the set operation in 2.
4. Perform a vector set operation on the tile pairs retrieved in 3.

Weiler's graph comparison algorithm is used to perform the desired set operation for each pair of relevant tiles in step 4 above. The graph comparison algorithm yields a tile in the resultant hybrid structure. A node for the tessellation component of the resultant hybrid structure is constructed based on the contents of the resultant tile and the location and sizes of the input tiles. Since the input hybrid structures may have different tile decompositions, the parallel traversal must keep track of the different sizes of tiles which may correspond to the same area.

The cost of the parallel traversal for a set operation is governed by the cost of attribute-based access described above. Thus, the cost of the traversal is $O(m_1+m_2)$ plus $O(w_1+w_2)$ retrievals, where m_1,m_2 are the number of nodes and w_1,w_2 are the number of relevant tiles in the input hybrid structures t_1,t_2, respectively. Since it is carried out in memory, on a tile pair basis, the cost of the vector set operation is governed by the cost of processing the vector data in the relevant tiles. The processing cost is $O(e_{w_1}e_{w_2})$ since all of the set operations involve intersection processing of every edge with every other edge, where e_{w_1},e_{w_2} are the number of edges in the relevant tiles of the two themes.

4.3.3. Thematic Overlay

Like the attribute-based operations, thematic overlay takes as parameters two hybrid structures and yields a resultant hybrid structure. The thematic overlay operation consists of a parallel traversal of the input structures with all tiles retrieved, regardless of color content. Each tile of one theme is overlayed by all of the tiles of the other theme which overlap the same area. Each tile pair overlay is performed using Weiler's graph comparison algorithm. Note that while some tiles, from either theme, may participate in more than one overlay operation in memory, all tiles are retrieved only once.

We now present a cursory cost analysis of an average case thematic overlay operation to illustrate the divide-and-conquer aspect of the hybrid model. To do this, we make some additional assumptions. As before, we assume that two themes, t_1 and t_2, partition the area of interest by polygon networks. The themes are described by e_1 and e_2 edges, respectively. Now, however, we also assume that the density of spatial data across the area of interest is uniform (i.e., the density of edges and density of polygons). That is, it is assumed these are well-behaved thematic data sets.

The polygon networks are represented by hybrid structures using the tile area resolution threshold. While the tile area threshold yields tiles of fixed area, the uniformity assumption implies that the tiles are of bounded capacity. Consequently, we assume that the tile decomposition for each theme yields m tiles. For large values of e_1, e_2 and a small value of m, we expect that each tile contains approximately e_i/m edges, where $i=1,2$ corresponding to the two themes.

The cost analysis involves a thematic overlay of the two themes. The hybrid model provides for a divide-and-conquer approach to solving the problem. The tile partition represents a way to subdivide the overlay problem into m smaller problems of a similar nature. Independent and local processing of each pair of tiles (one tile from each theme) is the key to the conquer aspect of the approach. A key feature of this approach is that the results of processing the tile pairs do not have to be combined to produce a final result.

We begin by considering the retrieval cost of performing an overlay operation with a totally vector representation (e.g., Weiler's data structure). In this case, overlay is performed on a polygon pair basis (i.e., each polygon of one theme is intersected with each polygon of the other theme). Effectively, this consists of intersecting each edge of theme t_1 with every edge of theme t_2. With enough memory, the simplest way to do this is to read both themes entirely into memory and conduct the intersection processing. Since we are dealing with very large themes, we assume that there is insufficient memory to store all e_1 and all e_2 edges simultaneously. With a totally vector representation, the retrieval cost is given by

$$\mathrm{R_v} = e_1 + \frac{e_1 e_2}{c}, \tag{4.1}$$

where c is a measure of the proportion of edges that may be stored in memory simultaneously. The first term of Equation 4.1 is for the retrieval of all edges of t_1, while the second term corresponds to the retrieval of all edges of t_2 for each edge of t_1. This retrieval cost is $O(e_1 e_2)$. In the extreme case of the ability to store only two edges at once, the retrieval cost is exactly $\mathrm{R_v} = e_1 + e_1 e_2$. As c increases, the cost $\mathrm{R_v}$ decreases to the limit of $\mathrm{R_v} = e_1 + e_2$, wherein all edges may be stored simultaneously in memory.

The hybrid approach subdivides the overlay problem into m overlays of tile pairs. Since the contents of two tiles can be stored in memory, all of the edges comprising the polygons and partial polygons contained in two tiles need be retrieved only once to perform the overlay of that tile pair. The retrieval cost for one tile pair is then $\mathrm{R_h} = e_1/m + e_2/m$. Since there are m such tile pairs, the retrieval cost for the overlay operation with the hybrid structure is given by

$$R_h = m \left[\frac{e_1}{m} + \frac{e_2}{m} \right] = e_1 + e_2. \tag{4.2}$$

In essence, use of the hybrid model implies that all of the edges in both themes are retrieved only once which gives a retrieval cost of $O(e_1+e_2)$. The divide-and-conquer strategy reduces the retrieval cost of overlay by an order of magnitude, that is, from quadratic cost to linear cost with respect to the number of edges.

We now consider the computation effort of the overlay operation. The fundamental task in overlay is the discovery and computation of the intersections that exist in a set of polygons, or specifically, in the set of edges comprising the polygons. Given a set of e edges, this task consists of processing every possible pair of edges in the set. With a totally vector representation such as Weiler's data structure, the processing cost of overlaying the two themes t_1, t_2 is

$$C_v = e_1 e_2, \tag{4.3}$$

since each edge of theme t_1 is tested against all of the edges of theme t_2. This processing cost is $O(e_1 e_2)$.[7]

The totally vector approach may be viewed as an overlay operation on two tiles with a total of e_1+e_2 edges. The hybrid approach subdivides this overlay into m pairs of tiles with each pair containing $e_1/m + e_2/m$ edges. The processing cost of the hybrid approach is given by

---

[7] The processing cost of overlay can be as low as $O(e \log e + s)$, where $e = e_1+e_2$ and s is the total number of intersections, by using data structures such as the k-structure [77] and the plane sweep algorithms of Nievergelt and Preparata [60]. However, use of these methods does not affect the cost advantage gained from applying the hybrid approach, as described below.

$$C_h = m \left[\frac{e_1}{m} \frac{e_2}{m} \right] = \frac{e_1 e_2}{m}. \tag{4.4}$$

This processing cost is also $O(e_1 e_2)$. Note that the processing cost of the hybrid approach is reduced from the cost of the totally vector approach by a factor of m. This result is worth noting for two reasons. First, subdivision of the problem by tiles reduces the processing cost for any number of overlay operations for the one-time expense of building the hybrid structures. Second, there is no recombination cost after all of the m overlays are completed. Each pair of tiles, once retrieved and processed, produces a corresponding tile in the result hybrid structure.

We now compare the cost of overlay between the hybrid model and a totally tessellation approach (a region quadtree). The overlay of the two themes represented by region quadtrees consists of retrieving corresponding nodes of the tessellations. The retrieval cost for this is $O(n_1+n_2)$ where n_i is the number of nodes of each region quadtree. The processing cost of a region quadtree overlay is also $O(n_1+n_2)$ since all corresponding nodes must be compared. However, the cost of node comparisons is insignificant in relation to the retrieval cost. Hunter [37] has shown that $n_i = O(e_i)$, as the resolution of the tessellation gets finer, where $i=1,2$. Consequently, the retrieval and processing costs are both also $O(e_1+e_2)$, where e_i is the number of edges in each theme. This statement has two implications. First, it implies that the retrieval cost of overlay with region quadtrees is of the same order as the cost with the hybrid model. Second, the processing cost of overlay with region quadtrees is an order of magnitude less than the cost with the hybrid model. Note, however, that in practice the region quadtree requires more storage than a vector representation. That is, $n_i \geq ce_i$, for some constant $c>1$. There is an interesting research problem regarding the performance tradeoff and storage tradeoff between a totally tessellation data structure and our hybrid data structure. Actual values for the constant c and the constants associated with $O(n_1+n_2)$ and $O(e_1+e_2)$ will enable a meaningful comparison.

4.3.4. Summary

We now summarize and compare the retrieval and processing costs of the hybrid model with totally tessellation and totally vector approaches. The presentation is organized by spatial operation.

The point inclusion operation can be performed with $O(1)$ retrievals using the hybrid approach. This is as good as the $O(1)$ retrievals required by a bucket method implementation of a tessellation approach. However, the hybrid approach requires additional vector processing to determine the enclosing polygon. The additional processing cost is $O(e_w)$, where e_w is the number of edges in the relevant tile. There is no processing overhead for a tessellation approach. In contrast, a vector approach to point inclusion requires retrieval and testing of all polygons in a theme. The totally vector cost is thus $O(e)$ retrievals and $O(e)$ processing.

In terms of windowing, the hybrid approach requires retrieval of only the relevant tiles. This compares with a tessellation approach in which only the relevant nodes (buckets) need be retrieved. Typically, however, there are many more relevant nodes in a region quadtree than in the tessellation component of the hybrid structure for a given theme. The hybrid approach requires additional vector processing within the relevant tiles while the tessellation approach merely requires simple comparisons. Windowing with a totally vector representation requires exhaustive retrieval and processing of all edges. The hybrid approach is clearly superior to this since only relevant tiles are retrieved and only the edges within the relevant tiles need be processed.

If storage space is not an issue, then the representation offering the best location-based performance is tessellation. The cost with the hybrid representation is only slightly more expensive since vector processing is localized to relevant tiles. Compared to vector, the hybrid approach offers superior performance at a slight increase in storage space. Therefore, considering both space and time complexity, the hybrid approach offers an effective compromise: near-tessellation performance with near-vector space requirement.

For the attribute-based operations, a tessellation encoding of each theme requires exhaustive traversal of both structures. This is an $O(n_i)$ retrieval cost, where n_i is the number of nodes in each theme. In contrast, using the hybrid structure for each theme, only the tessellation components must be exhaustively traversed to discover the relevant tiles. This is $O(m_i)$ where m_i is the number of tiles for each theme. The key difference is that typically, the number of tiles in a hybrid representation of a theme is much smaller than the number of nodes in a tessellation representation of the theme. Furthermore, only the relevant tiles need be retrieved and processed using the hybrid approach. Tiles that do not contain relevant colors need not be considered beyond the tessellation component traversal. In terms of processing, the hybrid approach requires vector set operations which are more costly than the comparison operations required with a tessellation representation.

With the hybrid approach, retrieval of relevant tiles for the attribute-based operations does not mean that only relevant polygons are retrieved. Each relevant tile may contain irrelevant polygons. In contrast, totally vector representations of polygon networks may be organized specifically by color. This means that the attribute-based operations may be resolved by retrieving only relevant polygons. In this case, the hybrid approach compares unfavorably with a vector approach. Note however, that the methods to improve attribute-based performance for tessellation methods discussed in Section 2.6 may be used to improve the performance of the hybrid model in this regard.

The representation offering the best attribute-based performance is vector. The cost with the hybrid representation is somewhat more costly since there is the potential for retrieval and processing of irrelevant vector data. Compared to tessellation, the hybrid approach offers superior performance due to reduced retrieval cost. Considering both space and time complexity, the hybrid approach offers an effective compromise: near-vector performance with near-vector space requirement.

Finally, the thematic overlay operation requires exhaustive retrieval and processing of entire themes regardless of the representation used. The hybrid model combines the advantages of both tessellation and vector methods. Retrieval cost is kept to linear order, like a tessellation method, but the amount of data involved is substantially less. The quadratic processing cost of the vector data is improved over a totally vector approach.

Table 4.1 provides a condensed version of the above comments. Note that in the tessellation column of Table 4.1, we ignore processing costs since there is one comparison per retrieval, which is insignificant.

Operation	Representation Method		
	Tessellation	Vector	Hybrid
Location-based	Retrieve only relevant nodes.	Exhaustive retrieval and processing of edges.	Retrieve only relevant tiles. Exhaustive processing within tiles.
Attribute-based	Exhaustive retrieval of nodes.	Retrieve and process only relevant edges.	Exhaustive retrieval of tessellation component. Retrieve and process only relevant tiles.
Thematic overlay	Exhaustive retrieval of nodes (linear order).	Multiple retrieval and processing of edges (quadratic order).	Exhaustive retrieval of tiles (linear order). Multiple processing of edges within tiles (improved quadratic order).

Table 4.1 Summary of Retrieval and Processing Costs.

4.4. Tile Methods

During the last twenty years, but especially in the last five to ten years, there has been considerable work in the development of GIS's using what we designate as *tile methods*. Tile methods resemble our hybrid model in that they use a tiling of the plane to subdivide space into cells. This organization usually serves as an index to spatial data stored in an underlying data structure, which is quite often a vector scheme. This section begins by reviewing several GIS applications that employ a tile method and compares them to our hybrid model. The comparisons are based on how the design aspects of these tile methods differ from our hybrid model. There are three aspects that appear repeatedly in our discussion. First, some tile methods do not partition the underlying vector data structure. Second, other methods partition the underlying structure but do not perform spatial operations in a tilewise manner. The third aspect is related to the manner in which these tile methods handle attribute-based retrieval. This section ends with a discussion of hybrid methods that do not resemble our hybrid model.

Canada Geographic Information System

The Canada Geographic Information System (CGIS) was probably the first GIS developed [86]. CGIS is intended for the storage and analysis of resource related thematic data as well as production of maps. The main data division of CGIS is the theme (coverage), which comprises data of a single descriptive variable, and can potentially cover all of Canada. While CGIS does partition the locational data representing a theme, it differs from our hybrid model in that it does not perform spatial operations in a tilewise manner.

The data structure for each theme has two-components - the Image Data Set (IDS) and the Descriptor Data Set (DDS). The IDS encodes boundary data for polygon networks representing themes while the DDS stores classification (attribute) data for each polygon. The IDS is partitioned by a tiling of the plane consisting of fixed area tiles, called *frames*, which are squares in the geodetic coordinate system (i.e., the boundaries of frames are based on and are parallel to latitude and longitude lines). The smallest tile used in CGIS is called a *unit frame*. An actual fixed tile size used for a theme is specified by the user and must be of unit frame size or larger. That is, CGIS corresponds to our hybrid model with a fixed tile area threshold. Larger frames are constructed by hierarchically combining groups of four unit frames in a manner resembling quadtrees. The frames of the IDS partition the vector data describing a theme making the IDS analogous to the vector component of our hybrid model. The unit frames are numbered and stored using the Morton encoding sequence in a sequential file organization. Therefore, the IDS also corresponds to the tessellation component of our hybrid, but it lacks attribute information. In CGIS, attribute information is stored instead within the DDS.

Within each IDS frame, a polygon-driven vector format is used to encode a theme. This format uses three variable length records in a polygon, chain, point hierarchy similar to POLYVRT. Each polygon is listed in a record ordered by polygon number and a pointer is used to point to the opening vertex of the first chain defining the polygon. All chains defining the polygon network are listed once in a chain record ordered by opening vertices. For each chain, there are pointers to the next chains defining the left and right polygons, the opening and closing nodes, and a pointer to the start of the vertex data for the chain. The vertex data for all chains in a frame consists of chaincodes stored sequentially in a *compact notation record*. Polygons represented within CGIS are partitioned by the boundaries of the IDS frames. Pseudo-edges are indicated and stored in the chain record for each frame. Polygons that cross frame borders have the same polygon number within each frame.

The attribute data for the polygons in the IDS is maintained separately in the DDS which is not partitioned by frames. Rather, the DDS for each theme consists of variable length records, ordered by polygon number, which store attribute data for each unpartitioned polygon in the theme. The data stored for each polygon includes its thematic attribute, area and centroid, and a list of the IDS frames that it intersects. Ostensibly, this makes the DDS an inverted list organization for attribute data. The DDS is ordered by polygon number which means that it must be exhaustively searched for operations requiring specific attributes. This is not a serious problem since an additional level of organization can be introduced to permit organization of the DDS based on attribute.

CGIS provides a thematic overlay function that facilitates the intersection or union of two to eight themes. Windowing can be performed in either of two ways: by frame number and by overlay with a polygonal window defined in a separate IDS. The overlay function operates on a set of entire themes each defined by an IDS/DDS pair. An overlay operation begins by merging the frames in each theme using an edge matching procedure that removes frame borders [69]. The overlay operation then proceeds on the merged polygon networks. The result of an overlay operation is a new IDS (which implies repartitioning of the overlayed polygons) and a corresponding DDS created by merging the original DDS's. While the partitioned organization of CGIS is similar in spirit to our hybrid model, the partition of themes imposed by IDS frames is not exploited for the tile-wise processing of overlay operations.

Cook [18] describes a geographic database, based on CGIS, that more closely resembles our hybrid model. Like CGIS, the main concept of Cook's structure is a partitioned organization of a vector encoded polygon network. Rather than using fixed area tiles, Cook's structure uses fixed capacity tiles (i.e., fixed size buckets). Bucket overflow is handled by splitting tiles into four quadrants like a quadtree. The system maintains a direct correspondence between a tile and its bucket by storing the buckets in Morton order. Cook states that access to the tiles is by coordinate-derived keys, presumably meaning that locational codes are used.

Cook's system can be used to retrieve tiles in the same manner as our hybrid for location-based operations. His system is more like our hybrid model than CGIS since it exploits the partitioned organization of the vector data for performing spatial operations. In particular, he describes algorithms for the tilewise processing of the windowing and point inclusion operations. A significant difference between Cook's system and both CGIS and our hybrid is that Cook advocates merging of multiple themes within one structure whereas CGIS and our hybrid use one structure per theme.

Like CGIS, Cook's system also organizes attribute data in a separate structure ordered by polygon. Among the items stored for each polygon are thematic attribute and the coordinates of a point within the polygon. The coordinates are used to retrieve the tile containing the polygon. Whereas CGIS stores attribute information for each unpartitioned polygon in a theme, Cook's system stores the attribute information for each partial polygon introduced by the partition. Nevertheless, Cook's system also requires exhaustive search for attribute-based access. Another difference between Cook's system and our hybrid is that our tessellation component is used for attribute search which is much smaller than Cook's polygon-driven attribute structure.

The ARC/INFO GIS

ARC/INFO is a well known GIS produced by the Environmental Systems Research Institute (ESRI) [58]. The system is capable of maintaining a cartographic database known as a *map library*. ESRI describes the map library as based on a hybrid data model representing locational and thematic (attribute) data. A vector model is used for locational data while a relational database model is used for the attribute data, hence ESRI's definition of hybrid is not the same as ours. Since ARC/INFO is designed as a tile method, it is included in this discussion. A major difference between ARC/INFO and our hybrid model is again that ARC/INFO allows for partitioning of locational data but does not employ tilewise processing of spatial operations.

The basic unit of operation is the *coverage* which is analogous to a mapsheet (a tile in our case) in conventional cartography [58]. A coverage represents both locational and thematic data for a given area in terms of features. Features include spatial entities such as nodes, arcs (chains), and polygons, all of which have locations and possibly attributes. For each coverage, ARC/INFO uses a polygon-driven vector method for the locational data. The structure used is a triplet of tables defining node, chain, and polygon topology in a fashion similar to POLYVRT [27]. Thematic data for chains and polygons is stored in feature attribute tables and managed by a relational database management system. Each record in these tables is keyed by chain or polygon identifier and contains one or more attributes of the feature.

ARC/INFO is designed to facilitate a number of polygon set operations on entire individual coverages with no provision for attribute restrictions. However, the relational database allows the attribute information associated with the vector data to be extracted by specific attribute (e.g., color). This is used to construct new coverages for set operations with attribute restrictions. The user has the option of performing these operations on individual coverages or the user may merge coverages and then process the merged set. The tilewise approach to set operations is not inherent in the design of ARC/INFO.

ARC/INFO provides management software called LIBRARIAN [27] that can be used to organize coverages into a map library. The organization takes place along two dimensions - by content into *layers* or themes and by location into *tiles*. Tiles are used to subdivide the area of interest into a set of non-overlapping cells and while they are generally rectangular, tiles may be of any shape. A spatial index, called the *index coverage*, is maintained for access to the tile structure. The index coverage is analogous to the tessellation component of our hybrid but lacks attribute information (as with CGIS). All locational information in a map library is partitioned by the tile structure [4, 58].

LIBRARIAN facilitates windowing on a collection of tiles in which the window may span one or more tiles. The extraction of a window is performed by defining the window as a polygon in a *selection coverage*. The relevant tiles are determined using the index coverage. These tiles are merged prior to window extraction by dissolving the tile boundaries and rebuilding the polygon network [7]. The window is then extracted with an overlay function. While the tile structure of the ARC/INFO map library serves as both a spatial index to and partition of the vector data, the design of ARC/INFO is not one of tilewise processing for spatial operations that span several tiles. Users of the system can operate in a tilewise manner if they choose, but ARC/INFO does not provide an integrated tilewise approach for the operations we are considering.

The CARIS GIS

The Computer Aided Resource Information System (CARIS) is used for maintaining land resource data and for the production of thematic maps [48]. The CARIS data structure consists of several files comprising a curve-driven vector scheme which is indexed by both a polygon table and a cell table. The cell table is a map of a square tiling of the plane that partitions a study area. Each record in the cell table stores pointers to the chains that are within or pass through the corresponding tile. The key difference between CARIS and the previous methods (CGIS, Cook, and ARC/INFO) is that the vector data structure referenced by the cell table is not partitioned by the tiling.

The central file in CARIS is the *descriptor file* which maintains the chains of a polygon network. Associated with each chain are two pointers for the left and right polygons and two pointers that designate one of the possible incident chains at each end of the chain. The function of the latter two pointers is similar to the threading technique used in the TIGER system in that they are used to retrieve the boundary of a polygon. In addition, they form a circular list identifying all neighbors of a polygon. A fifth pointer references the first element in a linked list of the edges that comprise the chain. Each edge element in the linked list contains a pointer to a record in a data file containing the coordinates of the edge.

The polygon table stores, among other data, two pointers per polygon. The first pointer references one of the component chains of a polygon in the descriptor file. A second pointer links a polygon with its attribute record in an attribute file. The attribute file is maintained by a commercial database management system. Note that the polygon table is not ordered by attribute (i.e., color).

The CARIS structure, as with POLYVRT and TIGER, is highly pointer based. The CARIS tiling does not partition the underlying vector data structure, rather it serves only as a spatial index. Location-based search is therefore localized to the cell level, but since pointers are used throughout the curve-driven vector structure, this can lead to a large number of retrievals to obtain relevant polygons. CARIS can be used for tilewise processing of spatial operations such as windowing and overlay. However, the vector data must then be multiply retrieved since it is not partitioned at tile boundaries. These operations are more costly compared to our hybrid approach for both retrieval and processing. Polygons spanning more than one relevant tile must be retrieved for each relevant tile and processing of one tile requires processing the vector data both within and outside the tile. For the attribute-based operations, CARIS requires exhaustive traversal of the polygon table to retrieve all polygons of a relevant color.

The GeoVision GIS

GeoVision Corporation [63] has developed a GIS for a range of applications including natural resources management. The spatial database of this GIS has two components - a spatial component which stores locational data for geographic entities and an attribute component which maintains descriptive and topological information of the entities in the spatial component. The GeoVision approach is similar to that of CARIS in that a spatial index is used to reference a polygon network. While CARIS uses an index based on fixed area tiles, the GeoVision system employs a quadtree index. Currently, one quadtree index is maintained for several themes. The resolution threshold governing the quadtree in the spatial component of the GeoVision system is the number of polygons threshold. Each leaf node of the quadtree stores a list of pointers referencing those polygons contained within or passing through the corresponding tile. Like CARIS, the underlying organization of the polygon network is not partitioned and appears to be encoded using a polygon-driven vector format.

The attribute component relates attribute data to polygons and is implemented by a relational database. The attribute component stores a *cell-feature table*, analogous to the quadtree index, which maintains quadtree cell addresses and pointers to the polygons that pass through the cells. Thus, a spatial index to the polygon network is maintained in both components of the system. The principal advantage of the cell-feature table is that it allows efficient retrieval of relevant vector data for attribute-based operations.

The GeoVision system supports both location-based and attribute-based access. A retrieval with both location and attribute restrictions proceeds as follows [36]. The overlap between a user specified window and the quadtree index is determined. The quadtree cells comprising this overlap are used to build a *cell table* that is joined with the cell-feature table in the attribute component to identify a set of candidate features. Attribute qualification within the relational database is used to refine the candidate feature set to only those features that possess the required attributes. The remaining candidate features are retrieved from the spatial component and their intersection with the query area completes the operation. When there is no attribute restriction to a retrieval, no cell table is built and the overlap between the quadtree and the window produces the candidate feature set directly. Conversely, a strictly attribute-based retrieval proceeds entirely within the attribute component.

Like CARIS, the GeoVision approach localizes location-based search to relevant cells. The GeoVision system requires multiple disk accesses to retrieve the polygons relevant to a tile. The quadtree index allows for tilewise processing of the location-based and overlay operations but the vector data is redundantly retrieved since it is unpartitioned. The attribute component effectively maintains an inverted list of the quadtree index indexed by attribute. This permits efficient attribute-based operations since only relevant vector data need be retrieved.

4.5. Other Hybrid Structures

Kleiner and Brassel [45] discuss storage strategies for spatial data on read-only external storage devices. The scope is a worldwide, multi-layered base for the environmental sciences comprised of zero to three dimensional spatial objects. Their main objectives are to minimize both access time and storage space. Spatial objects are distinguished between those that may be partitioned tiles, *background data*, and those that should be stored entirely within one bucket, *geographic object data*. This distinction is determined by the application and it is not clear what the value of such a separation might be in a thematic GIS context. Their proposal for storing background data is essentially a hybrid scheme. Space is hierarchically subdivided by a quadtree or bintree method into cells which are then ordered by locational code and managed with a B-tree. Each bucket contains partial background objects represented in a vector format. This method is not intended to allow for the reconstruction of the background objects nor the topology of the corresponding polygon network (in a two-dimensional setting).

To store geographic object data, Kleiner and Brassel propose that the Field Tree be used to index a set of buckets so as to avoid partitioning the objects. In a two-dimensional setting, each cell encodes a polygon network in a vector format and the entire structure is similarly organized as the background data structure. As previously discussed in Sections 2.6 and 3.5, there appears to be some benefit in using the Field Tree as the tessellation component for our hybrid model. However, the data-driven nature of the Field Tree, compared to other tessellation models that organize the embedding space, cause it to have more construction and updating overhead than the latter methods. The issues concerning the effectiveness of the Field Tree as a tessellation model and as the tessellation component for our hybrid model is an interesting topic for further research.

Langran and Buttenfield [47] advocate the concept of partitioning a spatial data set to enhance access and manipulability. They suggest systematic subdivision of files into small rectangular cells to support additive windowing. Ideally, the cell area should match that of the smallest window used. Additive windowing is performed by merging relevant cells rather than by clipping from larger tiles (i.e. subtractive). Additive windowing is cheaper than subtractive windowing if the relevant cells are merely aggregated rather than merged into a unified polygon network. While they discuss partitioning of spatial data, Langran and Buttenfield do not address the issue of using a tessellation structure to organize cells and provide for attribute-based access.

Other researchers suggest the use of clustering physical data storage according to the location of spatial data. This technique involves storing a spatial object (e.g., polygon) in its entirety within a bucket whose address is derived from the location of the object. For point data, the locational code is used to determine the address of the bucket. This is the basis of the cell methods discussed in Section 2.4. For spatial objects with extent, the average coordinate value of the describing vertices (e.g., centroid) or a corner of a bounding rectangle may be used to provide a locational code. The latter method is used by two developers of commercial GIS's. Keating, et al [43] use a hybrid of the R-Tree and the Field Tree for this technique. A bounding rectangle around an object is the basis for the location of the object. The object is stored in the smallest node (bucket) of a Field Tree that entirely contains the bounding rectangle of the object. Apparently, the object to bucket addressing is achieved by using locational codes. Bundock [8] describes a similar approach and implies the use of a Field Tree technique.

The main problem with the above clustering approach is the relationship between the contents of a bucket and the portion of a polygon network corresponding to the bucket. A bucket effectively corresponds to a tile, since a range of coordinate values hash to it. The problem is that the objects stored in the bucket are not partitioned at the tile boundary. Thus, a bucket may store vector data that is outside the extent of the corresponding tile. The retrieval and processing costs of location-based access is then increased since irrelevant vector data is retrieved. Conversely, vector data that is within a tile may be stored in buckets other than the bucket corresponding to the tile. Again, location-based retrieval cost is increased since multiple buckets may have to be retrieved for a given area.

A very different type of hybrid data structure has been proposed by Peuquet [66] as a solution to the problem of the storage and manipulation of very large spatial data sets.

The structure is called a *vaster* and uses a rectangular tiling to partition an area of interest. Each tile, known as a *swath*, is a rectangle of constant height (in y) and spans the width of the area. The number of swaths required to cover the area is determined by the height of the swaths. Each swath corresponds to a set of contiguous scan lines as occurs in a raster representation. The swaths are effectively a one-dimensional tessellation and they each contain a raster and vector component. Both components are encoded at a common resolution.

The raster component of a swath consists of recording the intercepts of partial chains that cross the leading boundary of the swath. This encodes the first scan line and serves as an index for all of the vector data in the swath. This index contains an identifier and x-coordinate for each partial chain. The chains are represented with a chaincode format and are stored in index scan line intercept sequence. Polygons contained entirely within a swath are stored separately.

The vaster does not have as effective a localizing property as our hybrid model. Location-based access to the swaths of the vaster is limited to one-dimension (i.e., by y-coordinate only), which is not as effective as a quadtree index. Rectangular swaths are not as effective as square tiles since, for applications of very large extent, the swaths may be very long. The vector component of each swath is essentially a spaghetti format, which is not as effective as the more sophisticated polygon-driven vector formats. The vaster has no provision for attribute-based access to the vector component.

Chapter 5

Concluding Remarks

5.1. Summary

The major contribution of this thesis is the introduction of a general hybrid data model for the representation of thematic data. The strength of the hybrid model is that it permits the classification of almost all specific, spatial data structures discussed in the literature. Limiting cases of the hybrid model correspond to the two fundamental models of spatial data representation, namely, the tessellation and vector models. Intermediate formulations of the hybrid model, arising from a combination of a resolution threshold and a decision made at the pixel level, yield specific hybrid data structures.

The classification of spatial data structures is organized according to three models. The first two are fundamental models, the location-based and attribute-based models. Location-based and attribute-based data structures are discussed in Chapters 2 and 3, respectively. The third model is a hybrid of the location-based and attribute-based models. Hybrid data structures are classified in Chapter 4.

The classification is also based on the evaluation of the effectiveness of data structures for the performance of spatial operations. The underlying assumption of this thesis is that of very large thematic data sets which must reside on secondary storage. Thus, the retrieval cost of accessing relevant thematic data for both location-based and attribute-based operations is of primary concern. The cost of processing thematic data in memory is given secondary consideration.

With an appropriate resolution threshold, an intermediate formulation of the hybrid model compares favorably with either a totally tessellation or a totally vector approach to representing thematic data. For location-based operations, the hybrid data structure is as cost effective as a tessellation approach. For attribute-based operations, the hybrid data structure offers near vector performance. For thematic overlay, the hybrid data structure offers an effective compromise: the linear retrieval cost of a tessellation structure and an improvement on the processing cost of a vector structure.

The hybrid data structure is also proposed as a superior practical alternative for the representation of very large thematic data sets in GIS applications. In terms of storage space, the hybrid data structure is only slightly larger than a vector data structure. As mentioned above, the hybrid data structure offers cost effective performance of the spatial operations.

There are several examples of implemented GIS's that employ data structures which resemble intermediate formulations of the hybrid model. As discussed in Section 4.4, these systems have one or more deficiencies when compared to the hybrid model. This thesis makes a case for a tiling or partitioning approach to organizing thematic data (other advocates include Cook [18] and Burrough [10]). However, there are proponents of the opposite approach, that is, a non-partitioned or *seamless* organization of thematic data. Among others, Bundock [8], Frank [31], and Kleiner and Brassel [45] advocate the storage of unpartitioned polygons within a set of fixed-size secondary storage buckets. While one or more polygons may be stored in their entirety in a bucket, the polygons comprising a polygon network are still distributed among the set of buckets. This is a logically seamless approach up to the polygon level. The CARIS GIS [48] and the GeoVision GIS [63] take the seamless concept further by storing an entire polygon network in one, unpartitioned data structure. The developers of these GIS's claim that the seamless approach offers performance benefits over a partitioned approach.

As mentioned above, this thesis demonstrates that, for attribute-based operations, a partitioned organization of vector encoded thematic data is almost as effective as a seamless (i.e., totally vector) approach. For the location-based operations and thematic overlay, the partitioned approach is superior to a seamless approach. A well-designed implementation of the hybrid structure for a GIS will yield results indistinguishable from a seamless approach, for the spatial operations considered in this thesis.

For applications in which location-based operations are dominant, the location-based data structures are clearly cost effective. Conversely, for applications performing primarily attribute-based operations, the attribute-based data structures are appropriate. It is possible to introduce additional structures to either location-based or attribute-based data structures for the purpose of enhancing the performance of those operations for which they are less effective. For example, the forest of quadtrees [41] is a storage effective method of introducing attribute indices to region quadtrees (which are location-based) to enhance attribute-based operations (see Section 2.6). Conversely, the attribute-based TIGER data structure [56], enhances location-based operations by the addition of a node table, ordered by Peano key (see Section 3.5), to its chain file. The hybrid data structure of this thesis provides similar enhancements and is effective for applications that perform both location-based and attribute-based operations.

5.2. Future Perspectives

During the course of this thesis, several issues and data structures were encountered that appear to be interesting topics for further research.

The Field Tree [31] is an interesting variation on the decomposition principle of the region quadtree. The splitting of quadrants combined with the shifting of levels tends to reduce the partitioning of small polygons across subtrees. This may be useful for enhancing attribute-based search with a tessellation structure. Conversely, the tendency of storing polygons within the smallest cells that completely enclose them, may enhance location-based search with the Field Tree used as a spatial index to vector data. Note, however, that the discussion in Section 2.2 suggests otherwise. The use of the Field Tree as a structure for a GIS has been suggested recently [8, 43, 45]. The viability of the Field Tree as either a GIS data structure or as the tessellation component of the hybrid model has yet to be fully explored.

Another structure that appears attractive for the realization of linear quadtrees or the tessellation component of the hybrid is the grid file [61]. The size and growth of the grid file and the simplicity of its table look-up access method make it attractive compared to the EXCELL method. Like EXCELL, the grid file does not require overflow buckets, whereas methods based on linear hashing generally do. The grid file appears unsuitable for the representation of a linear quadtree since the decomposition it induces is irregular. Requiring the decomposition to be regular might be an effective way to adapt the grid file to linear quadtrees.

The suitability of the k-structure [44] for the representation of a polygon network is interesting. Although the k-structure is elaborate and requires complete reconstruction for updating, it enables logarithmic retrieval cost for the performance of the point inclusion and thematic overlay operations; that is, $O(log\ e)$ and $O(e \log e)$ operations, respectively, where e is the total number of edges in the polygon network [77]. This is considerably better than either the BSPR or the strip tree. These last two structures offer linear retrieval cost for the point inclusion operation and quadratic retrieval cost for thematic overlay, with respect to e. Because of this, the k-structure should be investigated further as a location-based data structure for thematic data. Recent work by Elcock, et al [25] describes related triangular decompositions that may be applicable to polygon networks.

An open research problem is the complexity of spatial data. The following questions give a flavor of the issues involved in this area. How do we characterize non-uniformly distributed thematic data? Hunter has shown an asymptotically linear relationship between the size of a quadtree and the perimeter of a polygon (at a given resolution) [37]. What is the relationship between complexity of the data and the value of the constant in the asymptotic relationship? How is the complexity of the boundary of a polygon characterized or measured? The fractal nature of geographic lines [52] and complexity measures for polygon networks represented by quadtrees [19] are research areas that may yield results applicable to the design and application of data structures for non-uniformly distributed thematic data. Hunter's result also suggests an interesting research problem involving the relationship between the hybrid model and a tessellation model for the performance of the spatial operations (see Section 4.3.3).

The hybrid model proposed in this thesis suggests the parallel retrieval and processing of thematic data. This is possible since the spatial operations can be performed on a tilewise basis - each relevant tile (or pair of tiles) is independently retrieved and processed. Furthermore, there is no recombination cost since the results of the operations are represented by the hybrid model.

Parallel processing of tiles may be achieved by the current generation of multiple processor, parallel computers. Typically, these computers are designed for computationally intensive parallel algorithms that require very little initial data. In large GIS applications, the converse is typical, that is, large quantities of data must be retrieved and the processing costs are comparatively much lower. The benefits of adapting parallel processing to the hybrid model can only be realized if the data can be accessed in parallel.

Parallel retrieval of tiles implies the use of multiple storage devices that can be accessed simultaneously. These can be in the form of primary or secondary storage attached to a set of processors. The task is to distribute the tiles comprising a theme among the multiple storage devices. The question is, what is the best way to do this? Storing tiles in Morton order on external storage has the tendency to locate spatially nearby tiles physically close to each other. For operations such as windowing and overlay, the next tile to retrieve and process is often near the current tile. If spatially nearby tiles are located on the same storage device then they must be retrieved and processed serially. Since the hybrid model facilitates independent retrieval and processing of tiles, it seems apparent that scattering spatially nearby tiles on different storage devices might allow parallel retrieval and processing.

The windowing operation provides a good example of the ideal situation for parallel retrieval and processing. Positioning a rectangular window anywhere within an area of interest specifies an arbitrary set of adjacent, relevant tiles. To obtain maximum parallel benefit, any set of adjacent or nearby tiles should be arranged such that each tile is always located on a device which does not contain any other tile in the set. Recall that linear hashing tends to scatter the nodes of a linear quadtree among a set of buckets as the structure grows. This property might be exploited to scatter the tiles of the hybrid structure among multiple storage devices.

The file organization scheme M-cycle allocation, introduced by Wu and Burkhard [92] (see Section 2.4) is relevant to a parallel variation of our hybrid model. Their work strongly suggests that using the hybrid structure with M-cycle allocation may yield maximum parallel benefit for the windowing and thematic overlay operations. The effectiveness of this approach for the attribute-based operations is an interesting research topic.

References

[1] D. J. Abel, "A B+ Tree Structure for Large Quadtrees", *Computer Vision, Graphics, and Image Processing,* Vol. 27, 1984, pp. 19-31.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[3] N. Ahuja, "On Approaches to Polygonal Decomposition for Hierarchical Image Representation", *Computer Vision, Graphics, and Image Processing,* Vol. 24, No. 2, 1983, pp. 200-214.

[4] P. Aronson and S. Morehouse, The ARC/INFO Map Library: A Design for a Digital Geographic Database, *Proc. Auto-Carto Six*, Ottawa, Canada, 1983, pp. 372-382.

[5] D. H. Ballard, "Strip Trees: A Hierarchical Representation for Curves", *Communications of the ACM* , Vol. 24, No. 5, 1981, pp. 310-321.

[6] J. L. Bentley, "Multidimensional Binary Search Trees used for Associative Searching", *Communications of the ACM* , Vol. 18, No. 9, 1975, pp. 509-517.

[7] C. Brown, Personal Communication, Environmental Systems Research Institute, Redlands, California, May 15, 1987.

[8] M. S. Bundock, An Integrated DBMS Approach to Geographical Information Systems, *Proc. Auto Carto 8*, Baltimore, Maryland, 1987, pp. 292-301.

[9] W. Burkhard, "Interpolation-Based Index Maintenance", *BIT,* Vol. 23, 1983, pp. 274-294.

[10] P. A. Burrough, *Principles of Geographical Information Systems for Land Resources Assessment,* Oxford University Press, Oxford, U.K., 1986.

[11] W. Burton, "Representation of Many-Sided Polygons and Polygonal Lines for Rapid Processing", *Communications of the ACM* , Vol. 20, No. 3, 1977, pp. 166-171.

[12] I. Carlbom, "An Algorithm for Geometric Set Operations Using Cellular Subdivision Techniques", *IEEE Computer Graphics and Applications,* Vol. 7, No. 5, 1987, pp. 44-55.

[13] S. K. Chang, N. Donato, B. H. McCormick, J. Reuss and R. Rocchetti, A Relational Database System for Pictures, *Proc. IEEE Workshop on Picture Data Description and Management*, Chicago, Illinois, 1977, pp. 142-149.

[14] S. K. Chang and T. L. Kunii, "Pictorial Data-Base Systems", *IEEE Computer*, Vol. 14, No. 11, 1981, pp. 13-21.

[15] Z. T. Chen, "Quad Tree Spatial Spectra --- Its Generation and Applications", *Proc. International Symposium on Spatial Data Handling*, Vol. 1, 1984, pp. 218-237.

[16] Z. T. Chen and D. Peuquet, Quad Tree Spatial Spectra Guide: A Fast Spatial Heuristic Search in a Large GIS, *Proc. Auto-Carto 7*, Washington, D.C., 1985, pp. 75-81.

[17] D. Comer, "The Ubiquitous B-Tree", *ACM Computing Surveys* , Vol. 11, No. 2, 1979, pp. 121-137.

[18] B. G. Cook, "The Structural and Algorithmic Basis of a Geographic Data Base", *Proc. First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, Vol. 4, 1978, Harvard University.

[19] E. Creutzburg, "Complexities of Quadtrees and the Structure of Pictures", *Journal of Information Processing and Cybernetics (EIK)*, Vol. 18, No. 9, 1982, pp. 507-522.

[20] J. Dangermond, A Classification of Software Components Commonly Used In Geographic Information Systems, *Proc. U.S.-Australia Workshop on the Design and Implementation of Computer-Based Geographic Information Systems*, Honolulu, Hawaii, 1982, pp. 70-91.

[21] W. A. Davis, Data Formats for Geographical Data Bases, Final Report for Alberta Energy and Natural Resources, University of Alberta, Edmonton, Canada, 1984.

[22] W. A. Davis and C. H. Hwang, File Organization Schemes for Geometric Data, Dept. of Computing Science Tech. Rep. 85-14, University of Alberta, Edmonton, Canada, 1985.

[23] W. A. Davis, Hybrid Use of Hashing Techniques for Spatial Data, *Proc. Auto Carto London*, London, England, 1986, pp. 127-135.

[24] W. A. Davis and C. H. Hwang, Organizing & Indexing of Spatial Data, *Proc. Second International Symposium on Spatial Data Handling*, Seattle, Washington, 1986, pp. 5-14.

[25] E. W. Elcock, I. Gargantini and T. R. Walsh, "Triangular Decomposition", *Image And Vision Computing*, Vol. 5, No. 3, 1987, pp. 225-231.

[26] Energy and Natural Resources (now Forestry, Lands and Wildlife), 1:20,000 Digital Topographic Data Base, System External Specifications, Province of Alberta, 1983.

[27] Environmental Systems Research Institute, *ARC Users Manual (April 1985), ARC/INFO Programmers Manual (October 1985), LIBRARIAN Users Manual (July 1987)*, ESRI, Redlands, California.

[28] R. Fagin, J. Nievergelt, N. Pippenger and H. R. Strong, "Extendible Hashing - A Fast Access Method for Dynamic Files", *ACM Trans. on Database Systems*, Vol. 4, No. 3, 1979, pp. 315-344.

[29] C. Faloutsos, T. Sellis and N. Roussopoulos, Analysis of Object Oriented Spatial Access Methods, *Proc. ACM SIGMOD Conference on Management of Data*, 1987, pp. 426-439.

[30] A. U. Frank, Application of DBMS to Land Information Systems, *Proc. 7th International Conference on Very Large Data Bases*, Cannes, France, 1981, pp. 448-453.

[31] A. U. Frank, Storage Methods for Space Related Data: The FIELD TREE, Bericht Nr. 71, Eidgenossische Technische Hochschule (ETH), Zurich, Switzerland, 1983.

[32] A. U. Frank, "Requirements for Database Systems Suitable to Manage Large Spatial Databases", *Proc. International Symposium on Spatial Data Handling*, Vol. 1, 1984, pp. 38-60.

[33] H. Freeman, "Computer Processing of Line-Drawing Images", *ACM Computing Surveys* , Vol. 6, No. 1, 1974, pp. 57-97.

[34] I. Gargantini, "An Effective Way to Represent Quadtrees", *Communications of the ACM* , Vol. 25, No. 12, 1982, pp. 905-910.

[35] A. Guttman, R-Trees: A Dynamic Index Structure For Spatial Searching, *Proc. ACM SIGMOD Conference on Management of Data*, Boston, Mass., 1984, pp. 47-57.

[36] O. Halustchak, Personal Communication, GeoVision Corporation, Ottawa, Canada, Dec. 16, 1986.

[37] G. M. Hunter and K. Steiglitz, "Operations on Images Using Quad Trees", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 1, No. 2, 1979, pp. 145-153.

[38] C. H. Hwang, File Organization Schemes for Geometric Data, M.Sc. Thesis, University of Alberta, Edmonton, Canada, 1985.

[39] C. H. Hwang and W. A. Davis, A File Organization Scheme for Polygon Data, *Proc. Graphics Interface '86*, 1986, pp. 287-292.

[40] International Business Machines Corporation, MVS/Extended Architecture JCL Reference, GC28-1352, IBM, 1987.

[41] L. Jones and S. S. Iyengar, "Space and Time Efficient Virtual Quadtrees", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 6, No. 2, 1984, pp. 244-247.

[42] E. Kawaguchi and T. Endo, "On a Method of Binary Picture Representation and its Application to Data Compression", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 2, No. 1, 1980, pp. 27-35.

[43] T. Keating, W. Phillips and K. Ingram, "An Integrated Topologic Database Design for Geographic Information Systems", *Photogrammetric Engineering And Remote Sensing*, Vol. 53, No. 10, 1987, pp. 1399-1402.

[44] D. Kirkpatrick, "Optimal Search in Planar Subdivisions", *SIAM J. Comput.*, Vol. 12, No. 1, 1983, pp. 28-35.

[45] A. Kleiner and K. E. Brassel, Hierarchical Grid Structures for Static Geographic Data Bases, *Proc. Auto Carto London*, London, England, 1986, pp. 485-496.

[46] K. Knowlton, "Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes", *Proc. of the IEEE 68*, Vol. 7, 1980, pp. 885-896.

[47] G. Langran and B. P. Buttenfield, Formatting Geographic Data to Enhance Manipulability, *Proc. Auto Carto 8*, Baltimore, Maryland, 1987, pp. 201-210.

[48] Y. C. Lee, A Data Structure for Resource Mapping With CARIS, *Proc. Auto-Carto Six*, Ottawa, Canada, 1983, pp. 151-160.

[49] W. Litwin, Linear Hashing: A New Tool for File and Table Addressing, *Proc. 6th International Conference on Very Large Data Bases*, 1980, pp. 212-223.

[50] R. A. Lorie and A. Meier, Using A Relational DBMS for Geographical Databases, Computer Science Research Report 3848 (43915), IBM Research Laboratory, San Jose, CA, 1983.

[51] G. Maffini, "Raster Versus Vector Data Encoding and Handling: A Commentary", *Photogrammetric Engineering And Remote Sensing*, Vol. 53, No. 10, 1987, pp. 1397-1398.

[52] B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman & Co., San Francisco, 1982.

[53] D. M. Mark and J. P. Lauzon, "Linear Quadtrees for Geographic Information Systems", *Proc. International Symposium on Spatial Data Handling*, Vol. 2, 1984, pp. 412-430.

[54] D. M. Mark and J. P. Lauzon, Approaches for Quadtree-Based Geographic Information Systems at Continental or Global Scales, *Proc. Auto-Carto 7*, Washington, D.C., 1985, pp. 355-365.

[55] J. J. Martin, Organization of Geographical Data With Quad Trees and Least Squares Approximation, *Proc. IEEE Conference on Pattern Recognition and Image Processing*, Las Vegas, Nevada, 1982, pp. 458-463.

[56] R. W. Marx, "The TIGER System: Automating the Geographic Structure of the United States Census", *Government Publications Review*, Vol. 13, 1986, pp. 181-201.

[57] R. D. Merrill, "Representation of Contours and Regions for Efficient Computer Search", *Communications of the ACM* , Vol. 16, No. 2, 1973, pp. 69-82.

[58] S. Morehouse, ARC/INFO: A Geo-Relational Model for Spatial Information, *Proc. Auto-Carto 7*, Washington, D.C., 1985, pp. 388-397.

[59] G. Nagy and S. Wagle, "Geographic Data Processing", *ACM Computing Surveys*, Vol. 11, No. 2, 1979, pp. 139-181.

[60] J. Nievergelt and F. P. Preparata, "Plane-Sweep Algorithms for Intersecting Geometric Figures", *Communications of the ACM* , Vol. 25, No. 10, 1982, pp. 739-747.

[61] J. Nievergelt, H. Hinterberger and K. C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure", *ACM Trans. on Database Systems,* Vol. 9, No. 1, 1984, pp. 38-71.

[62] J. A. Orenstein, Spatial Query Processing in an Object-Oriented Database System, *Proc. ACM SIGMOD Conference on Management of Data,* Washington, D.C., 1986, pp. 326-336.

[63] J. Palimaka, O. Halustchak and W. Walker, "Integration of a Spatial and Relational Database Within a Geographic Information System", *Technical Papers ACSM-ASPRS Annual Convention,* Vol. 3, 1986, pp. 131-140.

[64] T. Pavlidis, *Algorithms for Graphics and Image Processing,* Computer Science Press, Rockville, Maryland, 1982.

[65] T. Peucker and N. Chrisman, "Cartographic Data Structures", *The American Cartographer,* Vol. 2, No. 1, 1975, pp. 55-69.

[66] D. J. Peuquet, "A Hybrid Structure for the Storage and Manipulation of Very Large Spatial Data Sets", *Computer Vision, Graphics, and Image Processing,* Vol. 24, No. 1, 1983, pp. 14-27.

[67] D. J. Peuquet, "Data Structures for a Knowledge-Based Geographic Information System", *Proc. International Symposium on Spatial Data Handling,* Vol. 1, 1984, pp. 372-391.

[68] D. J. Peuquet, "A Conceptual Framework and Comparison of Spatial Data Models", *Cartographica,* Vol. 21, No. 4, 1984, pp. 66-113.

[69] B. Rizzo, Personal Communication, Canada Land Systems Division, Lands Directorate, Environment Canada, Mar. 9, 1987.

[70] A. Rosenfeld, H. Samet, C. A. Shaffer and R. E. Webber, Application of Hierarchical Data Structures to Geographical Information Systems, Computer Science Tech. Rep. 1197, University of Maryland, College Park, MD, 1982.

[71] A. Rosenfeld and A. C. Kak, *Digital Picture Processing,* Academic Press, New York, NY, 1982. Second Edition.

[72] A. Rosenfeld, H. Samet, C. A. Shaffer and R. E. Webber, Application of Hierarchical Data Structures to Geographical Information Systems Phase II, Computer Science Tech. Rep. 1327, University of Maryland, College Park, MD, 1983.

[73] N. Roussopoulos and D. Leifker, Direct Spatial Search on Pictorial Databases Using Packed R-trees, *Proc. ACM SIGMOD Conference on Management of Data*, Austin, Texas, 1985, pp. 17-31.

[74] R. D. Rugg, "Building a Hypergraph-Based Data Structure", *Cartographica*, Vol. 21, 1984, pp. 179-187.

[75] H. Samet, A. Rosenfeld, C. A. Shaffer and R. E. Webber, "Use of Hierarchical Data Structures in Geographical Information Systems", *Proc. International Symposium on Spatial Data Handling*, Vol. 2, 1984, pp. 392-411.

[76] H. Samet and R. E. Webber, "On Encoding Boundaries with Quadtrees", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 6, No. 3, 1984, pp. 365-369.

[77] H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys*, Vol. 16, No. 2, 1984, pp. 187-260.

[78] H. Samet and R. E. Webber, "Storing a Collection of Polygons Using Quadtrees", *ACM Trans. on Graphics*, Vol. 4, No. 3, 1985, pp. 182-222.

[79] H. Samet, C. A. Shaffer, R. C. Nelson, Y.-G. Huang, K. Fujimura and A. Rosenfeld, "Recent Developments in Linear Quadtree-Based Geographic Information Systems", *Image And Vision Computing*, Vol. 5, No. 3, 1987, pp. 187-197.

[80] M. Shneier, "Two Hierarchical Linear Feature Representations: Edge Pyramids and Edge Quadtrees", *Computer Graphics and Image Processing*, Vol. 17, No. 3, 1981, pp. 211-224.

[81] T. R. Smith and M. Pazner, "Knowledge-Based Control of Search and Learning in a Large Scale GIS", *Proc. International Symposium on Spatial Data Handling*, Vol. 2, 1984, pp. 498-519.

[82] M. Tamminen, Efficient Spatial Access to a Data Base, *Proc. ACM SIGMOD Conference on Management of Data*, 1982, pp. 200-206.

[83] M. Tamminen, "The Extendible Cell Method for Closest Point Problems", *BIT*, Vol. 22, 1982, pp. 27-41.

[84] M. Tamminen, "Performance Analysis of Cell Based Geometric File Organizations", *Computer Vision, Graphics, and Image Processing*, Vol. 24, No. 2, 1983, pp. 160-181.

[85] S. L. Tanimoto, Hierarchical Picture Indexing and Description, *Proc. IEEE Workshop on Picture Data Description and Management*, Asilomar, California, 1980, pp. 103-105.

[86] R. F. Tomlinson, H. W. Calkins and D. F. Marble, *Computer Handling of Geographical Data: An Examination of Selected Geographic Information Systems*, The Unesco Press, Paris, France, 1976.

[87] U.S. Department of Commerce, Bureau of the Census, The DIME Geocoding System, Report No. 4, Census Use Study, 1970.

[88] Various Authors, Session on Database Architecture, *Proc. Auto Carto 8*, Baltimore, Maryland, 1987, pp. 278-379.

[89] T. C. Waugh, A Response to Recent Papers and Articles on the Use of Quadtrees for Geographic Information Systems, *Proc. Second International Symposium on Spatial Data Handling*, Seattle, Washington, 1986, pp. 33-37.

[90] W. Weber, "Three Types of Map Data Structures, Their Ands and Nots, and a Possible Or", *Proc. First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, Vol. 4, 1978, Harvard University.

[91] K. Weiler, "Polygon Comparison Using a Graph Representation", *Proc. ACM SIGGRAPH 80 (pub. as Computer Graphics)*, Vol. 14, No. 3, 1980, pp. 10-18.

[92] C. T. Wu and W. A. Burkhard, "Associative Searching in Multiple Storage Units", *ACM Trans. on Database Systems*, Vol. 12, No. 1, 1987, pp. 38-64.

Appendix A1

Storage Requirement for the Hybrid Data Structure

This appendix is intended to give a quantitative example of the storage required by a hypothetical realization of the hybrid data structure proposed in Chapter 4. The purpose of this example is to illustrate the relative size of the tessellation component compared to the vector component.

We begin by describing an operational context. The department of Forestry, Lands and Wildlife of the Province of Alberta administers several digital mapping programs of provincial coverage and of varying scales. In particular, the Land Information Services Division of Forestry, Lands and Wildlife maintains a digital map data base at a scale of 1:20,000 [26]. This map series consists of 3030 maps, each with dimensions 0.250 degrees longitude by 0.125 degrees latitude (approximately, 16 km by 12 km). The 1:20,000 map series serves as a base for thematic applications which might employ the hybrid data structure.

We assume the existence of a theme that uses up to 750,000 bytes of storage per 1:20,000 map for vector encoded polygon networks. This is a substantial thematic data set which requires a total of about 2.3 gigabytes[8] (GB) of storage. If we subdivide each 1:20,000 map into 16 tiles, then each tile requires up to 46,875 bytes of storage.

For the purposes of this example, we assume that the data for this theme is stored on an IBM 3380K direct access storage device [40]. The pertinent characteristics of this device are listed in Table A1.1.

---

[8] Throughout this appendix, the terms *megabyte* and *gigabyte* refer to 10^6 and 10^9 bytes, respectively.

Characteristic	Value
Bytes/track	47,476
Tracks/cylinder	15
Cylinders/volume	2655
Bytes/volume	1.8907 GB

Table A1.1 Characteristics of the IBM 3380K Device.

A disk track can be viewed as a unit of storage that may be retrieved with a cost of one disk access. Each tile of the above theme may be stored entirely within one track of the IBM 3380K device. The entire theme can be stored on two 3380K devices. This organization allows location-based access to any tile in hybrid structure with $O(1)$ disk accesses.

The tessellation component must have 16 nodes for each 1:20,000 map (since there are 16 tiles) giving a total of 48,480 nodes. Each node consists of a 4-tuple of <key,level,color,data> (c.f. section 4.2). The storage required for each 4-tuple is shown in Table A1.2.

Element	Storage (bytes)
Key	16
Level	4
Color	40
Data	4
Total	64

Table A1.2 Storage Requirement for a Tessellation Component Node.

The key element in Table A1.2 allows for the storage of a locational code formed by interleaving the bits of two 8 byte coordinates. The level element consists of an integer level number. Use of 40 bytes for the color element allows up to 320 colors to be listed for each node. The data element consists of an integer pointer used to reference a specific cylinder and a specific track within the cylinder. The space required for all of the tessellation component nodes is thus 3.103 megabytes (MB).

The tessellation component is realized with the EXCELL cell method which requires a directory to index the nodes. If we assume a bucket capacity of $b=16$ tessellation component nodes, then the EXCELL directory consists of 3030 elements. Each directory element contains a 4 byte pointer which references a bucket. Thus, the space required for the directory is 0.012 MB.

The total space requirement for the tessellation component is 3.115 MB. The space required for the tessellation component is about 0.1% of the space required for the vector component. Thus, the hybrid realization of a large thematic data set introduces a minor storage overhead compared to a totally vector implementation.